WEB aplikacija za prikupljanje i analizu meteorolških podataka

Rašić, Nino

Master's thesis / Diplomski rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:565443

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-25**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli

Fakultet ekonomije i turizma

“Dr. Mijo Mirković”

NINO RAŠIĆ

WEB APLIKACIJA ZA PRIKUPLJANJE I ANALIZU METEOROLOŠKIH PODATAKA

Diplomski rad

Pula, srpanj 2018.

Sveučilište Jurja Dobrile u Puli

Fakultet ekonomije i turizma

“Dr. Mijo Mirković”

NINO RAŠIĆ

WEB APLIKACIJA ZA PRIKUPLJANJE I ANALIZU METEOROLOŠKIH PODATAKA

Diplomski rad

JMBAG : 0036447231, izvanredni student

Studijski smjer : Poslovna informatika

Predmet: Dinamičke web aplikacije
Znanstveno područje:
Znanstveno polje:
Znanstvena grana:

Mentor: prof.dr.sc. Mario Radovan

Pula, srpanj 2018.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani _____, kandidat za magistra _____ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, _____, _____ godine

IZJAVA

o korištenju autorskog djela

Ja, _____ dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom _____

_____

koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____ (datum)

Potpis

_____

SADRŽAJ

1. UVOD 1
1.1. Predmet i cilj rada 1
1.2. Izvori podataka i metode prikupljanja 1
1.3. Sadržaj i struktura rada 2
2. RAZVOJ WEB APLIKACIJA 3
2.1. Arhitektura web aplikacije 3
2.2. Razvoj web aplikacija 6
2.2.1. Model vodopada 7
2.2.2. Iterativni model 10
2.2.3. Agilna metoda razvoja 11
3. KORIŠTENE TEHNOLOGIJE KOD IZRADE WEB APLIKACIJE 14
3.1. Python 15
3.1.1. Primjeri popularnih Web stranica razvijenih u Python-u 17
3.2. Flask framework 18
3.3. Virtualna okruženja 20
3.4. Jinja 2 22
3.5. ORM i SQLAlchemy 23
3.6. Marshmallow 27
3.7. REST 29
3.8. PostgreSQL baza podataka 33
3.8.1. PostGIS 39
3.9. Redis - caching 39
3.10. JWT autentifikacija 42
4. WEB APLIKACIJA 43
5. ZAKLJUČAK 48
LITERATURA 49
POPIS SLIKA 51

Sažetak 54

Summary 55

1. UVOD

1.1. Predmet i cilj rada

Meteorološki podaci su od ogromne vrijednosti u današnjem vremenu i nalaze primjenu u gotovo svim sferama svakodnevnog života, od korištenja podataka za razvoj modernih meteoroloških prognostičkih modela do jedne od najčešćih tema razgovora. Iako je dosta mali dio zemlje pokriven meteorološkim mjernim instrumentima, svejedno postoji ogroman broj meteoroloških postaja koje svakodnevno prikupljaju podatke u rezoluciji od jednog dana pa čak do jedne minute. Time se generira velika količina podataka koju treba prikupiti i prikazati na jednostavan i efikasan način budući da podaci gube na vrijednost ako se nigdje ne prikazuju niti arhiviraju.

U radu će se obrađivati problematika vezana uz ovu temu tako da će se izraditi jednostavna web aplikacija koja će imati mogućnost primanja, prikazivanja i arhiviranja meteoroloških podataka koristeći Python kao glavni programski jezik, PostgreSQL bazu podataka te REST web servise.

1.2. Izvori podataka i metode prikupljanja

Za pisanje teorijskog dijela rada koristit će se domaća i strana literatura. Većina literature će biti strana literatura u obliku znanstvenih i stručnih članaka.

1.3. Sadržaj i struktura rada

Rad je strukturiran u pet glavnih cjelina. U uvodu se opisuje problem kojeg će se obrađivati u radu. U drugoj cjelini će biti riječi o općenitom načinu razvoja web aplikacija te o metodologijama koje se koriste. Treća cjelina govori o tehnologijama koje se koriste u praktičnom radu. Opis rezultata dobivenih u praktičnom radu se nalazi u četvrtoj cjelini nakon čega slijedi zaključak.

2. RAZVOJ WEB APLIKACIJA

Po definiciji, web aplikacija je aplikacija dizajnirana da joj se pristupa preko Interneta koristeći jedan od web preglednika[1] što je glavna razlika u odnosu na klasične aplikacije koje se pokreću preko operativnog sustava budući da se aplikacija ne mora razvijati za više platformi, nego joj može pristupiti svatko tko ima internetsku vezu. Time se ubrzava razvoj aplikacije i omogućava jednostavnija nadogradnja budući da se nadogradnje ne moraju prosljeđivati svakom pojedinom korisniku nego se izvrši samo na serveru i svi korisnici će odmah biti u mogućnosti koristiti najnoviju verziju web aplikacije. Također, svi podaci koji su obrađeni u web aplikaciji se pohranjuju na serveru čime se omogućava jednostavan pristup podacima s drugih uređaja te su često podaci pohranjeni na barem još jednoj dodatnoj lokaciji što korištenjem raznih strategija za "backup" podataka smanjuje mogućnost gubitka istih.

Naravno svaka nova tehnologija uz svoje prednosti nosi i nedostatke. Web aplikacije često imaju ograničen pristup sistemskim resursima zbog čega zahtjevnije aplikacije još uvijek često samo postoje kao obične aplikacije koje se mogu pokrenuti samo preko operativnog sistema. Također, javlja se i problem sigurnosti budući da sve spremamo na servere te time sigurnost podatka stavljamo u ruke kreatora aplikacije, što ponekad zna završiti izgubljenim ili kompromitiranim podacima.

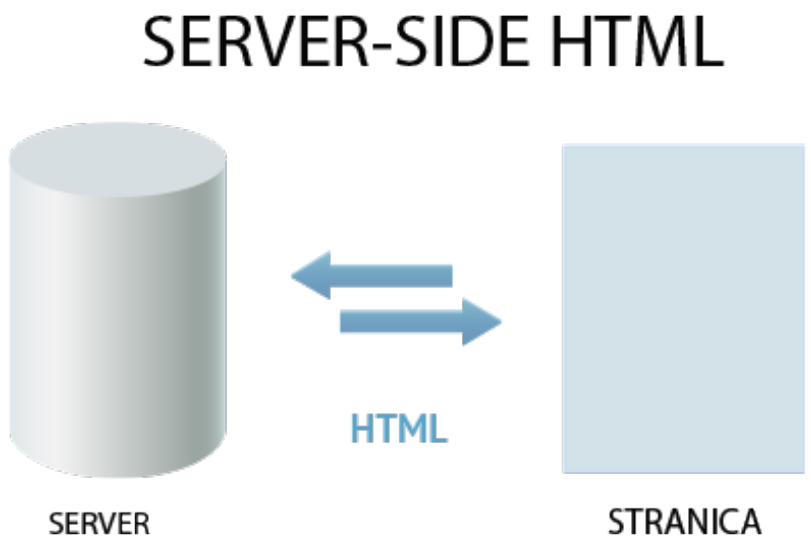
2.1. Arhitektura web aplikacije

Postoje tri glavne klijent-server arhitekture web aplikacija, svaka sa svojim prednostima i manama zbog čega postoji primjena za svaku od njih.

Najstarija i najraširenija arhitektura je "server-side HTML[2]" arhitektura u kojoj se HTML stranica u potpunosti generira na serveru te se dostavlja klijentu. Budući da je

[1] Oxford living dictionaries, dostupno na: https://en.oxforddictionaries.com/definition/us/web_app
[2] Hypertext Markup Language

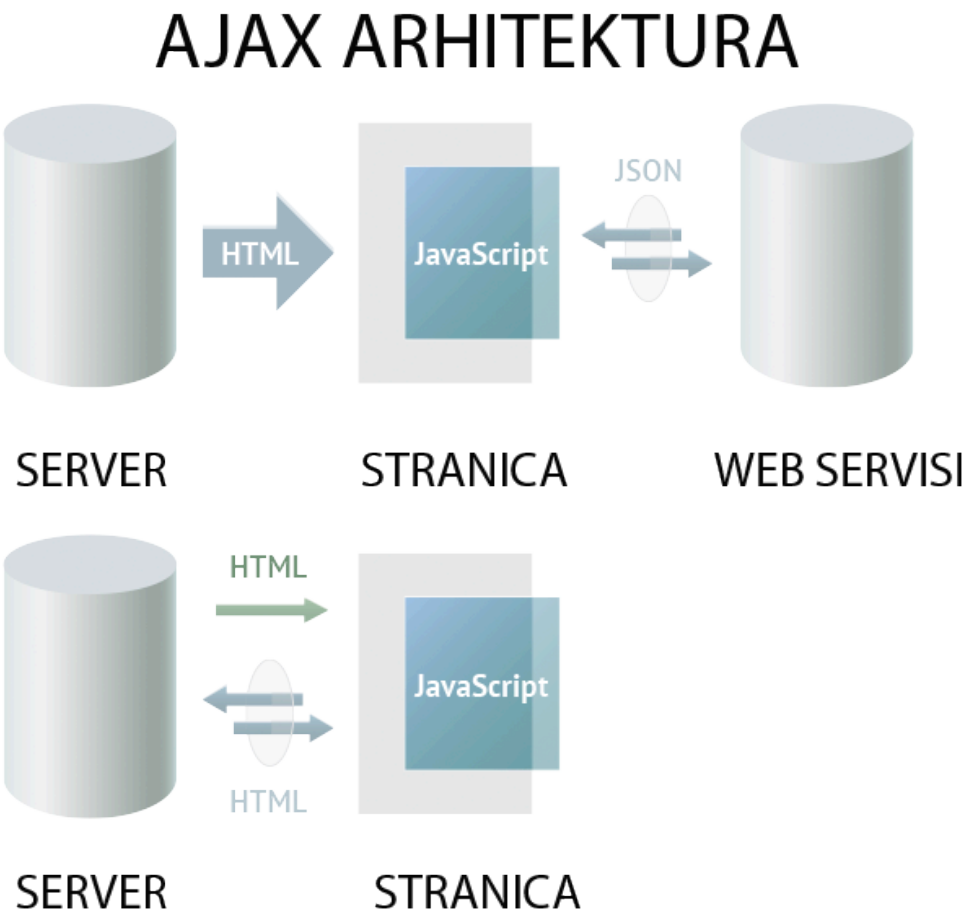
najjednostavnija za korištenje, dosta se početnika odlučuje za ovu arhitekturu za koju također ima i najviše materijala. Prednosti ove arhitekture su i te što je razvoj jednostavnih web aplikacija poprilično brz te se s jednostavnošću može servirati klijentu.



Slika 1. Prikaz server-side arhitekture
Izvor: https://mobidev.biz/blog/3_types_of_web_application_architecture

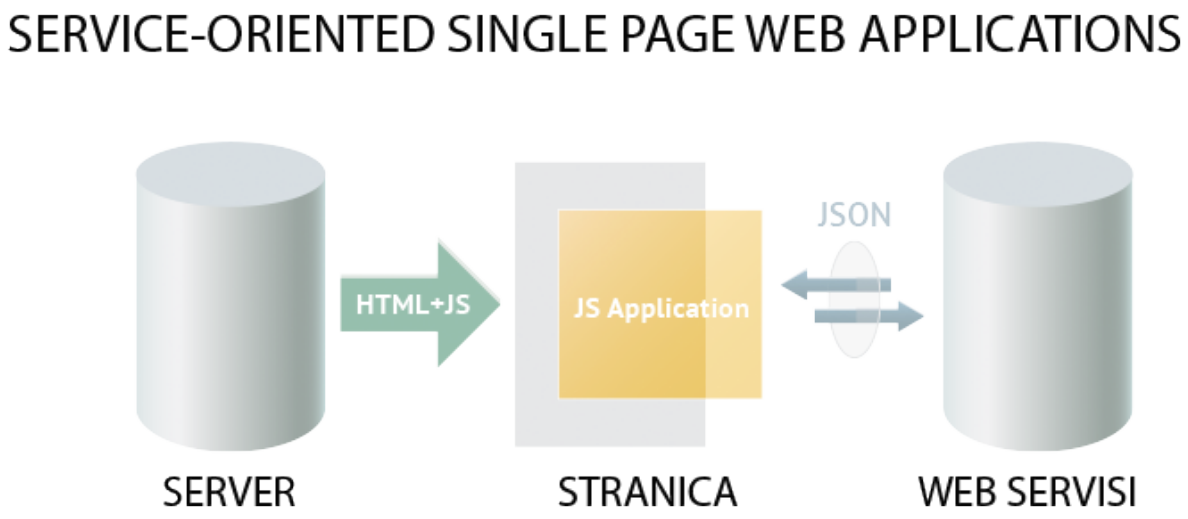
Nedostaci takve arhitekture su ti što se za svaku akciju, koliko god bila jednostavna, cijela stranica mora ponovno učitati i time se stvara velika količina podataka koji se prenose, narušava korisničko iskustvo te zapravo isključuje mogućnost bilo kakvog ”real-time” sustava te za takve sustave moramo koristiti naprednije arhitekture kao što je “Ajax”[3] arhitektura.

[3] Asynchronous JavaScript And XML



Slika 2. Prikaz AJAX arhitekture
Izvor: https://mobidev.biz/blog/3_types_of_web_application_architecture

Ajax sam po sebi nije vrsta tehnologije nego pristup razvoju web aplikacija koristeći određene vrste tehnologija (Slika 2). Te tehnologije nam omogućavaju da web aplikacija može prikazivati i slati nove podatke bez da se cijela stranica ponovno učitava, učitavajući samo dio koji nam je potreban čime se poboljšava korisničko iskustvo te smanjuje količina generiranog prometa.



Slika 3. Prikaz Service-orinted single page web applications arhitektura
Izvor: https://mobidev.biz/blog/3_types_of_web_application_architecture

Treća, najnovija arhitektura je “Single-page web application” (Slika 3) čime je generiranje HTML-a i gotovo sva poslovna logika premještena na klijenta čime se povećava responzivnost stranice, smanjuje količina generiranog prometa te se značajno smanjuje opterećenje sistemskih resursa budući da su serveri korišteni za rad s podacima, a ne za samo generiranje web aplikacija. Naravno, nije isključeno da server obavlja neke zadatke koji su njemu prikladniji kao što su generiranje datoteka ili izvršavanje zadataka čija bi izvedba trebala biti skrivena od javnosti.

2.2. Razvoj web aplikacija

Planiranje razvoja web aplikacija igra veliku ulogu u završnoj kvaliteti same aplikacije, što je aplikacija kvalitetnija to je veći postotak uspješnosti same aplikacije čime se povećava zadovoljstvo korisnika. Također, neplanirani pristup razvoja često dovodi do više cijene razvoja aplikacije što ponekad dovodi do toga da se aplikacija nikad ne razvije do kraja zbog visoke cijene neplaniranih troškova.

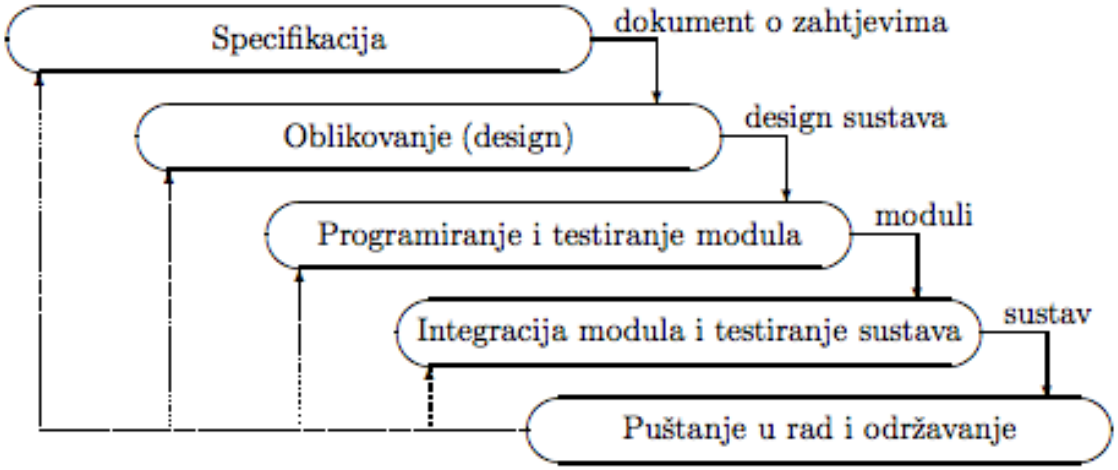
Razvoj web aplikacija je razvoj u različitim faza koje sadrže aktivnosti s namjerom za bolje planiranje i upravljanje. Postoji dosta razvojnih metoda aplikacija koje se koriste već duži niz godina kao što su model vodopada, spiralni model, iterativni model, evolucijski model, agilne metode i sl.

2.2.1. Model vodopada

Jedna od najpoznatijih i najčešće korištenih metoda kod razvoja aplikacija u kojoj se svi procesi specifikacije odvijaju slijedno.Takvim linearnim pristupom se omogućava razdvajanje projekta u smislene faze s točno definiranim ciljevima. Prvo se pojavljuje u građevinskoj industriji, a u nedostatku boljih metoda, pronalazi svoje mjesto i u razvoju aplikacija. Obično podrazumijeva 5 faza razvoja[4]:

- Faza analize specifikacija
- Faza oblikovanja
- Faza programiranja i testiranja modula
- Faza integracije modula i testiranja sustava
- Faza puštanja u rad i održavanja

[4] R.Manger, Softversko inženjerstvo (skripta), Prirodoslovno matematički fakultet u Zagrebu, 2005., str.3.



Slika 4. Model vodopada
Izvor: http://ttl.masfak.ni.ac.rs/SUK/Softversko%20inzenjestvo.pdf

1. Faza analize specifikacija

U ovoj fazi se skupljaju i dokumentiraju svi zahtjevi željene aplikacije. Potrebno je odvojiti dosta vremena za ovu fazu budući da strukturirana i dobro dokumentirana analiza može ubrzati razvoj aplikacije tako da prepozna sve moguće probleme i nađe moguća rješenja prije prelaska u sljedeću fazu.

2. Faza oblikovanja

Koristeći specifikaciju iz prve faze, radI se planiranje arhitekture sustava, prvih "prototipa" te kao rezultat toga, opisuje kako bi razvoj trebao izgledati s tehničke strane.

3. Faza programiranja i testiranja modula

Nakon što su rezultati iz faze oblikovanja prihvaćeni, kreće faza programiranja koja bi trebala teći nesmetano ako su prve dvije faze dovoljno kvalitetno napravljene, inače bi se mogli pojaviti prvi veći problemi vezani uz razvoj.

4. Faza integracije modula i testiranja sustava

Tijekom ove faze, odvija se testiranje i kontrola kvaliteta razvijene aplikacije. Tim stručnjaka izvršava kontrolu testirajući sve aktivnosti specificirane u prvoj fazi te se razvoj vraća na prethodnu fazu u slučaju pronalaska nekih nepravilnosti.

5. Faza puštanja u rad u održavanja

Zadnja faza razvoja u kojem se aplikacija pušta u rad te se osigurava podrška koja će služiti za održavanje i pravilno funkcioniranje aplikacije prilikom njezinog rada.

Prednosti modela vodopada su sljedeće:

- Strukturirani pristup
- Faze i aktivnosti su dobro definirane
- Pomaže kod planiranja i organiziranje projekta
- Svaka faza ima jasne ciljeve
- Široko prihvaćena u poslovnom svijetu

Mane modela vodopada su sljedeće[5]:

- Navedene faze u praksi je teško razdvojiti pa dolazi do naknadnog otkrivanja grešaka i vraćanja u prethodne faze

- Zbog tendencije da se zbog poštivanja rokova u određenom trenutku "zamrzne" pojedina faza, može se dogoditi da je sustav u trenutku puštanja u rad već neažuran i zastario

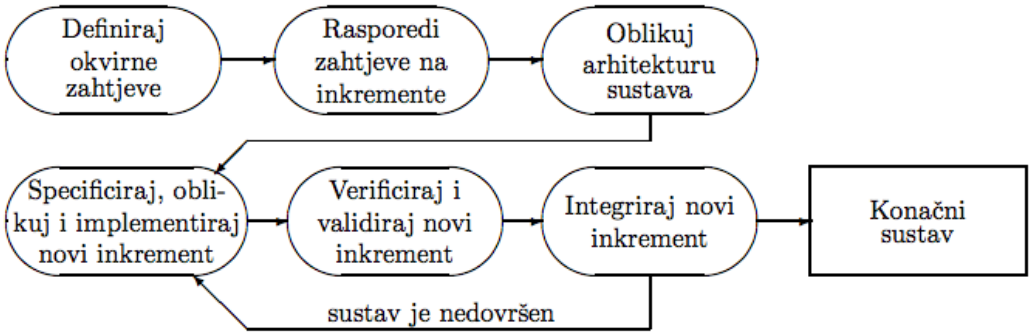
[5] ibidem, str. 3.

2.2.2. Iterativni model

Razvijen je da pokrije nedostatke modela vodopada tako da se razvija u nizu iteracija, razvijajući u svakoj iteraciji manji dio funkcionalnosti sustava nad kojim se radi verifikacija i validacija nakon čega razvoj prelazi u sljedeću iteraciju u kojoj se ne dotjeruje već realizirani dio sustava, nego mu dodaje sasvim novi dio[6]. Osnovna ideja za iterativnog razvoja je postupno razvijanje softverskog sustava na takav način da projektni tim može odmah iskoristiti iskustva iz prethodnih razvojnih faza .Iskustva se dobivaju kako tijekom razvoja tako i od korištenja već završenog dijela sustava.

Postupci u tom procesu su:

- Krenuti s jednostavnom i proširivom implementacijom definiranog podskupa zahtjeva sustava
- Postupno razvijanje s povećanjem znanja o zahtjevima sustava
- Prilagodba dizajna na svakoj iteraciji – razlikovanje između iteracija za implementaciju novih funkcija i iteracija za poboljšanje arhitekture sustava, odnosno refaktoring



Slika 5. Inkrementalni razvoj aplikacije
Izvor: http://ttl.masfak.ni.ac.rs/SUK/Softversko%20inzenjestvo.pdf

[6] Ibidem. str. 5

Prednosti modela inkrementalnog razvoja:

- Prve verzije aplikacije u samim počecima razvoja
- Fleksibilnost kod uvođenja novih zahtjeva
- Ranije uočavanje problema
- Prioritetni razvoj

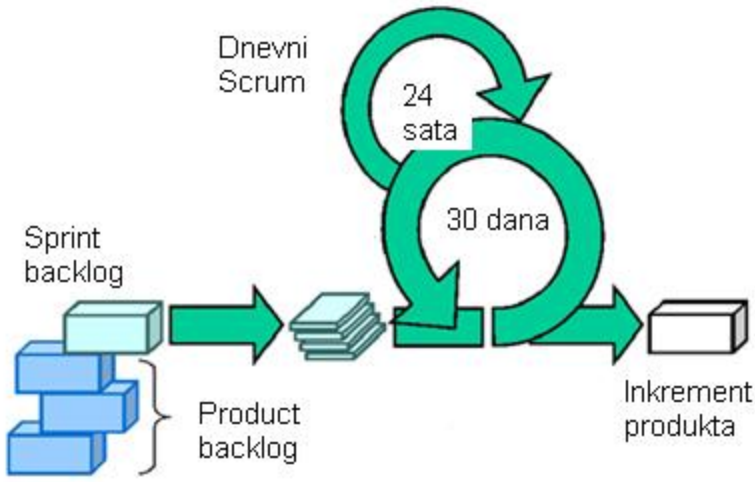
Mane modela inkrementalnog razvoja:

- Razdvajanje funkcionalnosti u smislene inkremente može biti kompliciran proces
- Povećani troškovi razvoja budući da arhitektura sustava nije strogo definirana i planirana na samom početku razvoja čime može doći do neplaniranih troškova

2.2.3. Agilna metoda razvoja

Bazirana na iterativnom i inkrementalnom razvoju gdje se zahtjevi i rješenja prilagođavaju tijekom međusobne interakcije između samoreguliranih i višenamjenskih timova. Zadnjih godina se često koristi, pogotovo u manjim start-up kompanijama koje si ne mogu priuštiti mjesece planiranja te kojima se specifikacije sustava učestalo mijenjaju ovisno o prioritetima.

Umjesto da se napravi detaljno planiranje na samome početku projekta, zahtjevi se mijenjaju tijekom trajanja projekta te se potiču stalne povratne informacije o korisnika. U agilnoj metodologiji se potiče komunikacije, timski rad te od odgovornost. Svi moraju raditi zajedno kako bi se uskladio proizvod s ciljevima kompanije te potrebama samih korisnika.



Slika 6. Agilna SCRUM metoda
Izvor: https://www.info-novitas.hr/o-nama/metodologije-rada/scrum-procesni-framework/

SCRUM je najpoznatija agilna metoda čiji je pristup empirijski, inkrementalan te iterativan. Temelji se na iskustvu da su mnogi razvojni projekti previše složeni da bi bili uključeni u cjeloviti plan. Bitan dio zahtjeva i pristupa rješenjima je nejasan na početku. Ova dvosmislenost može se ukloniti stvaranjem privremenih rezultata. Planiranje u SCRUM-u (Slika 6) je iterativno i inkrementalno, postoji *"Product backlogc* što je dugoročan plan koji se kontinuirano pročišćava i poboljšava, "*Sprint backlog"* što je detaljan plan koji nastaje samo za sljedeći ciklus koji može trajati od jednog dana od mjesec dana.

Prednosti agilne metode:

- Smanjuje se vrijeme do "izbacivanje" novih funkcionalnosti aplikacije
- Konstantna interakcija između klijenta, programera i testera čime se jamči kvaliteta aplikacije
- Fleksibilnost i jednostavna integracija novih funkcionalnosti

Nedostaci agilne metode:

- Teško je procijeniti potrebni uloženi trud kod većih aplikacija
- Nedostatak dokumentacije
- Nedostatak jasnog kraja razvoja aplikacije
- Prioritet su nove funkcionalnosti čime se stvara “tehnički dug” s kojim dolazi problem održavanje starih funkcionalnosti

3. KORIŠTENE TEHNOLOGIJE KOD IZRADE WEB APLIKACIJE

Za izradu web aplikacija postoji dosta različitih tehnologija koje se mogu odabrati. Često se te tehnologije dijele u dvije skupine, "*frontend*" i "*backend*". *Frontend* tehnologije koristimo za vizualni dio web aplikacije koji je dosta bitan budući da je to ono što korisnik vidi i što najviše utječe na korisničko iskustvo. U tu skupinu ulaze tehnologije kao što su HTML, Javascript i CSS

- HTML – sav kod u web aplikaciji se zapravo prevodi na kraju u HTML. To je jezik koje web preglednici razumiju i koriste ga za prikaz informacija korisniku

- CSS – za napraviti vizualno atraktivno aplikaciju nije dovoljno koristiti samo HTML budući da pruža samo osnovne stilske opcije i tu dolazi do izražaja CSS s kojim se može poboljšati prezentacija aplikacije s raznim stilskim opcijama

- Javascript – programski jezik koji se izvodi na klijentu te služi za kreiranje brzih intuitivnih korisničkih sučelja bez potreba za osvježivanjem aplikacije.

Backend tehnologije se nalaze na serveru te serviraju podatke *frontend*-u, najčešće putem REST-a[7]. U njima se izvršava poslovna logika aplikacije, rad s bazom podataka te raznorazni izračuni koji su potrebni web aplikaciji. Najrasprostranjenije tehnologije koje se koriste su Python, PHP i Ruby. U nastavku ovog poglavlja, opisat će se neke od tehnologija korištenih u praktičnom dijelu rada.

[7] Representational State Transfer

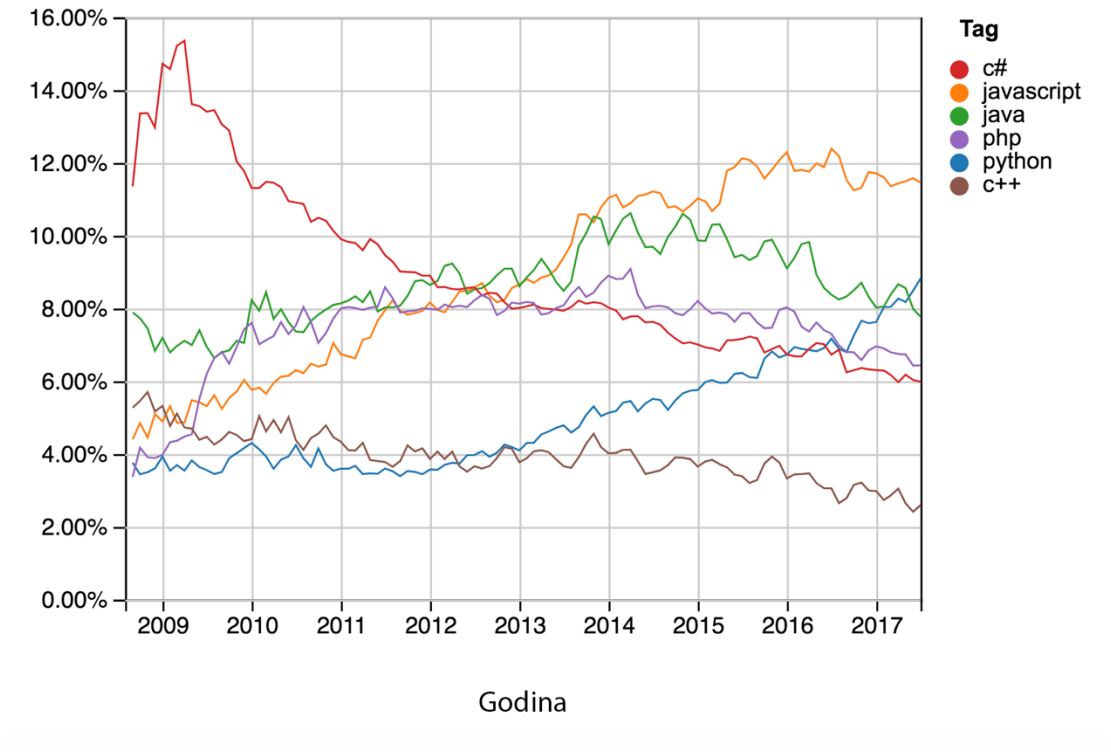
3.1. Python

Guido van Rossum je u ranim 1990. stvorio programski jezik Python kao nastavak na ABC programski jezik. Budući da je Python "*open source*" softver, trenutno je razvijan od strane programera širom svijeta koji se trude poboljšati i optimizirati njegove funkcionalnosti.

Python je zamišljen kao programski jezik opće namjene, interpretiran i visoke razine te je sličan programskim jezicima kao što su Perl, Ruby i sl [8]. Podržava nekoliko programskih paradigmi, kao što su objektno orijentirana, proceduralna, funkcionalna i imperativna te ima opsežnu standardnu biblioteku.

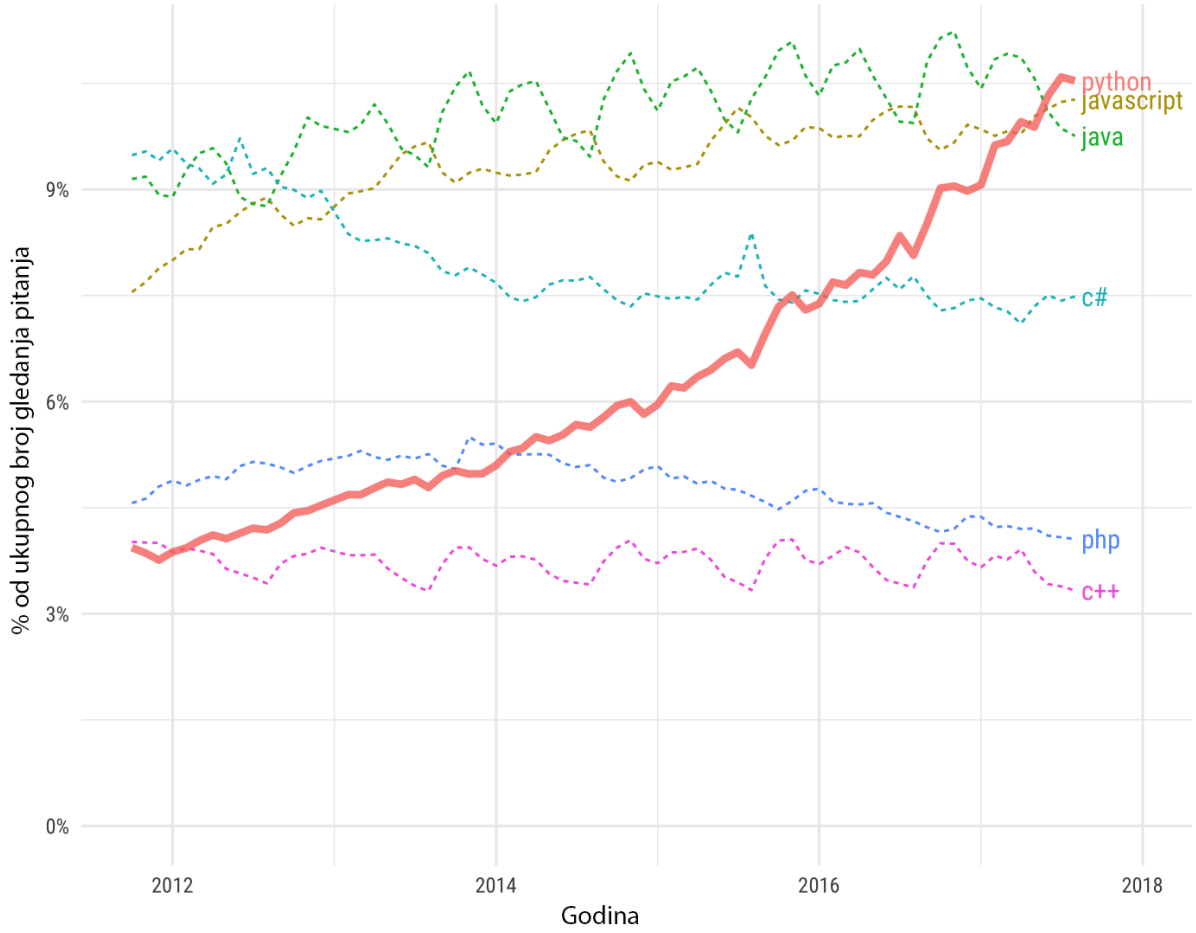
Zbog njegove jednostavnosti i brzine kojom je moguće razvijati aplikacije, postaje najrasprostranjeniji jezik koji se koristi za uvodna predavanja o programiranju na sveučilištima[9] te njegova popularnost raste iz godine u godinu (Slika 7. Postotak pitanja postavljenih o pojedinom programskom jeziku Izvor: https://stackoverflow.com).

[8] Wikipedia.org, https://hr.wikipedia.org/wiki/Python_(programski_jezik)
[9] StackOverflow, https://stackoverflow.blog/2017/09/06/incredible-growth-python/



Slika 7. Postotak pitanja postavljenih o pojedinom programskom jeziku
Izvor: https://stackoverflow.com

Posebno je zanimljiv porast postavljenih pitanja na stranici “Stack Overflow” u zemljama s visokim nacionalnim dohotkom (Slika 8) gdje je najtraženiji jezik što znači da se koristi u zemljama u kojima znanstvena istraživanja predstavljaju veliki udio u ekonomiji zemlje.



Slika 8. Postotak pitanja postavljenih o pojedinom programskom jeziku u zemljama s visokim nacionalnim dohotkom
Izvor: https://stackoverflow.com

3.1.1. Primjeri popularnih Web stranica razvijenih u Python-u

- YouTube – Jedna od najpoznatijih stranica na svijetu te najpoznatiji servis za dijeljenje video zapisa. Kada je stranica pokrenuta 2005., bila je bazirana na PHP-u, ali su godinu dana kasnije napravili migraciju na Python kojeg i dan danas koriste[10]

[10] Qura.com, https://www.quora.com/Which-Python-web-framework-was-YouTube-built-with-when-they-started-off

- Dropbox – servis za dijeljenje sadržaja koji nam omoguće pohranjivanje, sinkroniziranje te dijeljenje raznih dokumenata je također baziran na Pythonu. Dnevno se na Dropboxu pohrani preko milijardu dokumenata[11] što ga čini najvećih servisom za pohranu dokumenata

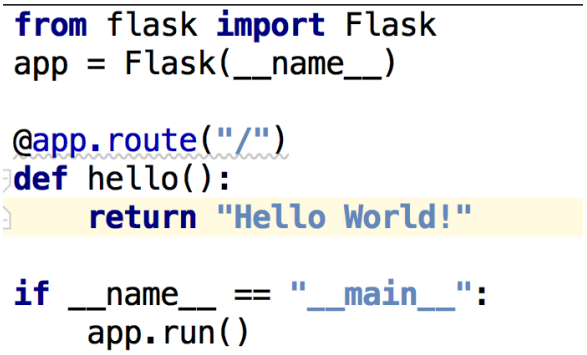
- Google – Python je uz Javu i C++, službeni jezik kompanije te se koristi u skoro svakom proizvodu te kompanije, počevši od same jezgre, a to su algoritmi korišteni u google tražilici

- Instagram – Web stranica/mobilna aplikacija namijenjena dijeljenju fotografija i video te jedna od najvećih društvenih mreža koristi Python za web servise

3.2. Flask framework

Flask je mali *framework* koji je baziran na Python programskom jeziku te je vrlo razumljiv i jednostavan za korištenje. Iako spada pod male *framework-e*, njegove mogućnosti su velike i čest je odabir kod razvoja web aplikacija, a koristit će se i za praktični dio ovoga rada.

[11] Expandedramblings.com, https://expandedramblings.com/index.php/dropbox-statistics/

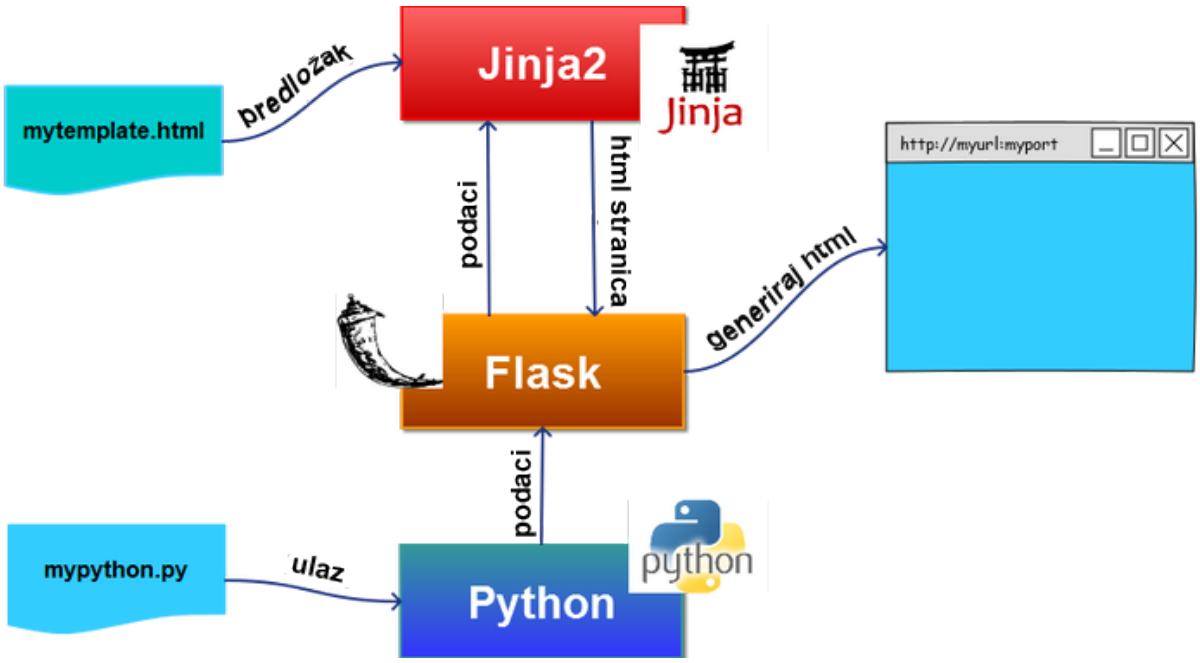


Slika 9. Primjer jednostavne Flask web aplikacije
Izvor: autor

Primjer jedne jednostavne Flask web aplikacije se može vidjeti na slici (Slika 9). Na toj slici se vidi kako je moguće u par linija koda dobiti potpuno funkcionalnu web aplikaciju s jednom rutom koja vraća “Hello World!” tekst na svaki poziv.

Flask je dizajniran tako da pruža osnovne usluge *framework-a* te je lako proširiv raznim nadogradnjama čime zapravo sam korisnik odabire što će koristiti u samoj aplikaciji te ne dolazi do nepotrebnog gomilanja koda. To je jedna od velikih prednosti Flask-a koju su prepoznali mnogi programeri koji žele razumljiv i čist kod. Nadogradnje postoje za razne funkcije kao što su za:

- Upravljanje autentifikacijom, *cookies, sessions*
- Korištenje predmemorije
- Lokalizacija
- Apstrakcijski sloj za baze podataka (ORM)
- Kompatibilnost s različitim bazama podataka



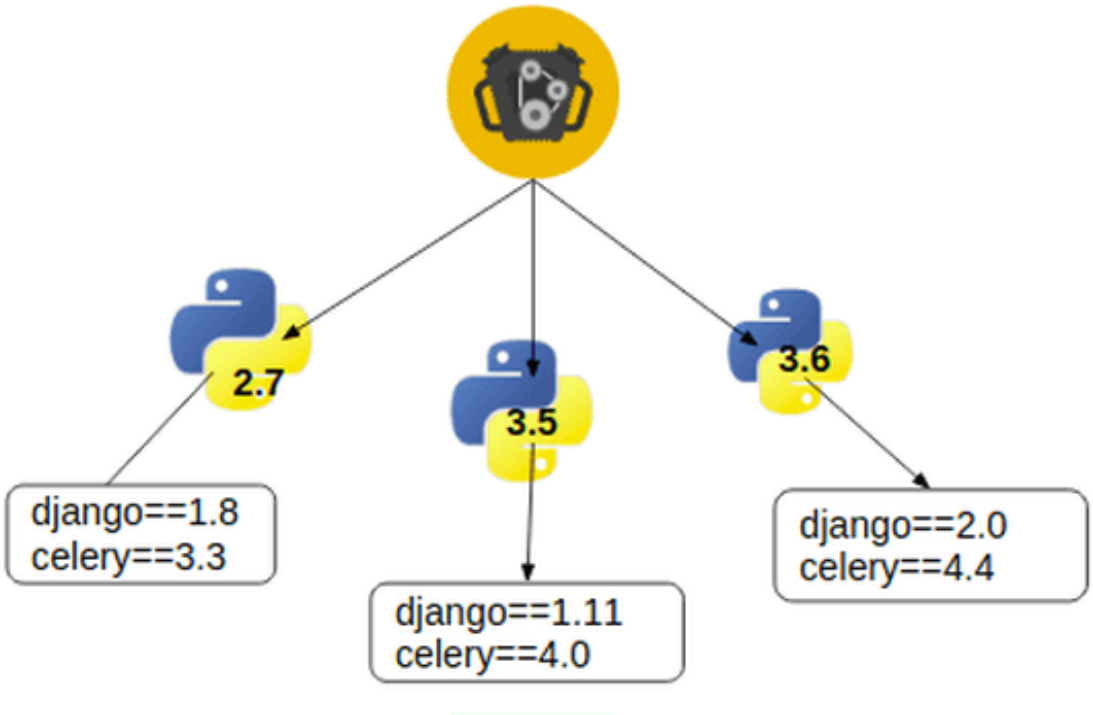
Slika 10. Prikaz rada Flask aplikacije
Izvor: https://gertjanvanhethofblog.wordpress.com/2016/02/10/exploring-flask-on-raspberry-pi/

U pozadini se Flask bazira na "Werkzeug" Python biblioteci te "Jijnja 2" *template engine-u*. Na slici (Slika 10) je prikazan rad Flask aplikacije, sve kreće od "mypthon.py" datoteke koja sadrži skriptu pisanu u Pythonu u kojem se koristi Flask biblioteka. Nakon što skripta kreira podatke koje bi htjeli prikazati u web aplikaciji, ti podaci se uz pomoć Flask biblioteke šalju u "Jijnja 2" gdje se uz pomoć predloška "mytemplate.html" generira *html* stranica koja se onda prikaže u web pregledniku.

3.3. Virtualna okruženja

Najpraktičniji način za razvijanje web-aplikacije, kao i zapravo svake Python skripte je taj da se koristi virtualno okruženje. Koristeći virtualno okruženje, stvara se kopija Python interpretatora u kojeg se mogu instalirati željeni paketi bez da se utječe na

globalni Python interpretator. Time se rješavaju raznorazni konflikti koji mogu nastati npr. prilikom razvoja nove skripte u kojoj se koristi najnovija verziju paketa dok druga skripta na sustavu nije kompatibilna s tom verzijom. Najčešći način korištenja je taj da se pomoću alata “virtualenv” ili “conda” napravi virtualno okruženje za projekt kojeg se želi napraviti. Nakon toga se instaliraju svi željeni paketi koji će se koristiti u projektu te se lista instaliranih paketa spremi u tekstualnu datoteku koja se onda može koristiti ako se želi replicirati virtualno okruženje na nekom drugom stroju. U praktičnom dijelu rada, cijela aplikacija se izvršava u virtualnom okruženju naziva “diplomski”.



Slika 11. Prikaz više virtualnih okruženja na jednom stroju
izvor: https://learnbatta.com/blog/how-to-install-python-virtualenv-19/

Na slici (Slika 11) je prikaz više virtualnih okruženja koja su instalirana na jednom stroju. Vidi se kako je moguće u svakom od okruženja imati različite verzije Python-a i instaliranih biblioteka te se olakšava upravljanje projektima.

3.4. Jinja 2

Jinja 2 je *template engine* koji služi da se uz pomoć tekstualne datoteke jednostavno generira odgovor (predložak) na poslužitelju kojeg se šalje klijentu. Evaluacija samog predloška se odvija unutar sigurnosne ograde (*Sandbox*) te se koristi sintaksa koja je slična sintaksi koja se koristi u Python-u.

```
<div class="card mb-4 box-shadow">
    <div class="card-header">
        <h4 class="my-0 font-weight-normal">Wind speed</h4>
    </div>
    <div class="card-body">
        <h1 class="card-title">{{current_wind_speed|safe}} <small class="text-muted">m/s</small></h1>
    </div>
</div>
<div class="card mb-4 box-shadow">
    <div class="card-header">
        <h4 class="my-0 font-weight-normal">Pressure</h4>
    </div>
    <div class="card-body">
        <h1 class="card-title">{{current_pressure|safe}} <small class="text-muted">hPa</small></h1>
    </div>
</div>
<div class="card mb-4 box-shadow">
    <div class="card-header">
        <h4 class="my-0 font-weight-normal">Relative humidity</h4>
    </div>
    <div class="card-body">
        <h1 class="card-title">{{current_relative_humidity|safe}} <small class="text-muted">%</small></h1>
    </div>
</div>
```

Slika 12. Primjer template-a koristeći Jinja 2
Izvor: autor

```
return render_template("station_data.html", chartID=chartID, chart=chart,
                       series=series, title=title, xAxis=xAxis,
                       yAxis=yAxis, page_title=page_title,
                       current_relative_humidity=current_rh,
                       current_temperature=current_temperature,
                       current_pressure=current_pressure,
                       current_wind_speed=current_wind_speed)
```

Slika 13. Evaluacija varijabli u Python-u te renderiranje predloška
Izvor: autor

. Na slici (Slika 12) se može vidjeti primjer predloška koristeći Jinja 2 *template engine.* Taj Jinja HTML predložak služi da bi se prikazali podaci o pojedinačnoj postaji.

U predlošku se dvostrukim vitičastim zagradama označavaju varijable koje bi programer htio da se evaluiraju koristeći Python. Na slici (Slika 13) je prikazan način renderiranja predloška u kojem ima funkcija “render_template” koja kao argumente prihvaća predložak koji se želi prikazati korisniku te sve podatke koji su nužni za renderiranje. U konkretnom slučaju, može se npr. vidjeti da se varijabla “current_relative_humidity” u predlošku postavlja na vrijednost “current_rh” varijable u funkciji za renderiranje predloška.

3.5. ORM i SQLAlchemy

SQLAlchemy je Python paket razvijen 2005. godine koji služi za interakciju s raznim bazama podataka kao što su PostgreSQL, MySQL i SQLite. Omogućuje kreiranje podatkovnih modela i SQL upita koristeći obične Python klase čime se olakšava rad s bazama podataka. Zbog takve enkapsulacije i zbog toga što se može koristiti s raznim bazama podataka, pojednostavljena je migracija sustava s jedne baze na drugu. Sam programer se ne mora brinuti o različitoj sintaksi među bazama te može biti siguran da će se svaki podatak unesen u bazu podataka pravilno sanitizirati.

SQLAlchemy pruža dva načina na koji se paket može koristiti, *SQL Expression Language* koji je tek blaga apstrakcija od klasičnih SQL upita, ali je standardiziran i pruža konzistentnost u radu s različitim bazama podataka te ORM[12] način koji služi da se upiti i manipulacija podataka vrše koristeći objektno-orijentiranu paradigmu te pruža visoku razinu apstrakcije.

Prednosti ORM-a su te što je podatkovni model na jednom mjestu te ga se može jednostavno održavati i nadograđivati, dosta toga je automatizirano te sam podatkovni model podataka nije toliko vezan za samu aplikaciju tako da ga je vrlo jednostavno upotrijebiti i u nekoj drugoj aplikaciji. Također, rad s ORM-om je vrlo prirodan te ubrzava razvoj aplikacije i pisanja upita na bazu.

[12] Object-relational mapping

Mana ORM-a su te što baza postaje previše apstraktna te sam programer ne zna uvijek što se točno događa u pozadini čime se rad sustava može usporiti i takvi upiti mogu postati vrlo neefikasni.

```python
class User(db.Model, BaseModel):
    """Users table"""

    __tablename__ = "users"

    first_name = db.Column(db.String(255))
    last_name = db.Column(db.String(255))
    username = db.Column(db.String(255), nullable=False)
    email = db.Column(db.String(255), nullable=False, unique=True)
    email_confirmed_at = db.Column(db.DateTime())
    password_hash = db.Column(db.String(255), nullable=False)
    active = db.Column(db.Boolean, default=False)

    roles = db.relationship("Role", secondary="user_roles")

    @property
    def password(self):
        raise AttributeError("Write only field")

    @password.setter
    def password(self, password):
        self.password_hash = Bcrypt().generate_password_hash(password).decode("utf-8")

    def password_is_valid(self, password):
        return Bcrypt().check_password_hash(self.password_hash, password)

    def has_roles(self, roles):
        for role in self.roles:
            if role.name in roles:
                return True
        return False

    @staticmethod
    def get_all():
        return User.query.all()

    @staticmethod
    def user_exists(self, user_data):
        if User.find_by_username(user_data["username"]) or User.find_by_email(user_data["email"]):
            raise KeyError("User already exists!")

    @staticmethod
    def find_by_username(username):
        return User.query.filter_by(username=username).first()

    @staticmethod
    def find_by_email(email):
        return User.query.filter_by(email=email).first()

    def __repr__(self):
        return "<User: {}>".format(self.username)
```

Slika 14. ORM podatkovni model User
Izvor: autor

Na slici (Slika 14) se može vidjeti ORM podatkovni model “User” koji nasljeđuje dvije klase, klasu “db.Model” i klasu “BaseModel” te služi za upravljanje podacima o korisniku. Na početku same klase se definira naziv tablice koja će se kreirati u bazi podataka za ovaj model te se kreće s definiranjem varijabli, odnosno stupaca u tablici “Users”. Za svaku varijablu se može definirati tip stupca, npr. varijabla “first_name” se definira kao tip *String* maksimalne dužine od 255 znakova dok se varijabla “active” definira kao tip *Booelan* te joj se može pridijeliti dva stanja, *True* ako je korisnik aktivan te *False* ako korisnik nije aktivan. Kod varijable “email” koju je definirana kao tip *String*, može se vidjeti da su odabrana još dva dodatna argumenta, prvi je “nullable” kojeg se postavlja na *False* te se time označava da vrijednost tog stupca ne može biti *null*, tj. uvijek se mora zadati neka vrijednosti kolumni “email”. Također se postavlja argument “unique” na *True* čime se označava da ne mogu postojati dvije jednake e-mail adrese u istoj tablici.

Naredbom “*roles = db.relationship("Role", secondary="user_roles")*” u podatkovnom modelu “User” se označava veza između tablice “users” te tablice “user_roles”. S ovim se postiže to da se s lakoćom dohvate sve uloge korisnika te na temelju toga se mogu odobriti ili zabraniti pristup određenim resursima.

Nakon toga su definirane dvije funkcije koje služe kao *getter* i *setter* enkapsulacijske metode za atribut “password”. Enkapsulacija se koristi zbog toga što šifra koju je unio korisnik prilikom registracije ne smije biti pohranjena kao čisti tekst u bazu podataka zbog sigurnosti podataka, nego se mora izvršiti *hash* funkcija nad tom šifrom. *Hash* funkcija nam omogućava da s lakoćom provjerimo da li je korisnik unio važeću šifru prilikom prijave na sustav, a onemogućavamo “curenje podataka” u slučaju da je baza kompromitirana.

Funkcija “get_all” je statička metoda kojom se dohvaćaju svi podaci o svim korisnike iz tablice “users”

Funkcija “find_by_username” služi za dohvaćanje korisnika sa zadanim korisničkim imenom, dok funkcija “find_by_email” dohvaća korisnika za zadanim e-mail-om.

```python
class WeatherStation(db.Model, BaseModel):
    """weather_stations table."""

    __tablename__ = "weather_stations"

    name = db.Column(db.String(255), nullable=False, unique=True)
    coordinates = db.Column(Geometry(geometry_type="POINT", srid=4326), nullable=False)
    owner_id = db.Column(db.Integer, db.ForeignKey("users.id"), nullable=False)
    above_sea_level = db.Column(db.Float)
    active = db.Column(db.Boolean, default=True)

    owner = db.relationship("User", backref="weather_stations")

    @staticmethod
    def get_all():
        return WeatherStation.query.all()

    @property
    def location(self):
        return self.coordinates

    @location.setter
    def location(self, point):
        lat, lon = point
        self.coordinates = "SRID=4326;POINT ({lon} {lat})".format(lon=lon, lat=lat)

    def __repr__(self):
        return "<Weather station: {}>".format(self.name)
```

Slika 15. ORM podatkovni model WeatherStation
Izvor: autor

Slika (Slika 15) prikazuje ORM podatkovni model “WeatherStation” koji služi za upravljanja podacima o meteorološkim postajama. Koristi tablicu “weather_stations” te ima stupac “name” tipa *String* za ime stanica, stupac “coordinates” koji je tipa *Geometry*. Za stupac “coordinates” određujemo postavljamo tip geometrije na točku te definiramo da će koordinate te točke biti u EPSG:4326 projekciji. U modelu imamo još stupce “owner_id” koji je strani ključ kojim se povezuje na tablicu “users” te dva stupca “above_sea_level” i “active”

```python
class WeatherData(db.Model, BaseModel):
    """weather_data table."""

    __tablename__ = "weather_data"

    station_id = db.Column(db.Integer, db.ForeignKey("weather_stations.id"), nullable=False)
    timestamp = db.Column(db.DateTime, default=db.func.current_timestamp())
    temperature_2m = db.Column(db.Float)
    wind_speed = db.Column(db.Float)
    wind_dir = db.Column(db.Float)
    mean_sea_level_pressure = db.Column(db.Float)
    relative_humidity = db.Column(db.Float)

    weather_station = db.relationship("WeatherStation", backref="weather_data")

    @staticmethod
    def get_all():
        return WeatherData.query.all()


    @staticmethod
    @cache.memoize(60*60)
    def get_n_latest_data_for_weather_station_id(id, n=100, schema=None):
        wd = WeatherData.query.filter_by(station_id=id).order_by("timestamp desc").limit(n).all()
        if schema:
            return schema.dump(wd, many=True).data
        else:
            return wd

    def __repr__(self):
        return "<Weather data: {}>".format(self.timestamp, self.temperature_2m)
```

Slika 16. ORM podatkovni model WeatherData
Izvor: autor

Slika 16 predstavlja ORM podatkovni model “WeatherData” koji uz već poznate funkcije sadrži “get_n_latest_data_for_weather_station_id” funkciju koja nam vraća *n* zadnjih meteoroloških zapisa za zadanu meteorološku stanicu.

3.6. Marshmallow

Marshmallow je ORM/ODM[13] biblioteka koja služi za konvertiranje kompleksnih tipova podataka kao što su objekti, u i iz nativnih Python tipova podataka[14], tj. služe za serijalizaciju i deserijalizaciju podataka.

[13] Object Document Mapper
[14] Django adapters documentation, https://media.readthedocs.org/pdf/django-adapters/latest/django-adapters.pdf

Za svaki ORM podatkovni model je potrebno napraviti shemu modela koje će služiti kao pomoći pri konvertiranju. Primjer jedne takve sheme može se vidjeti na slici (Slika 17).

```
class UserSchema(ma.ModelSchema):
    envelope_key = "users"

    class Meta:
        model = User

    _links = ma.Hyperlinks({
        "self": ma.URLFor("get_one_user", user_id="<id>"),
        "collection": ma.URLFor("get_user")
    })

    @validates("username")
    def validate_username(self, username):
        if User.find_by_username(username):
            raise ValidationError("Username already taken!")

    @validates("email")
    def validate_email(self, email):
        if User.find_by_email(email):
            raise ValidationError("Email already exists!")

    @post_dump(pass_many=True)
    def wrap(self, data, many):
        key = self.envelope_key
        return {
            key: data
        }
```

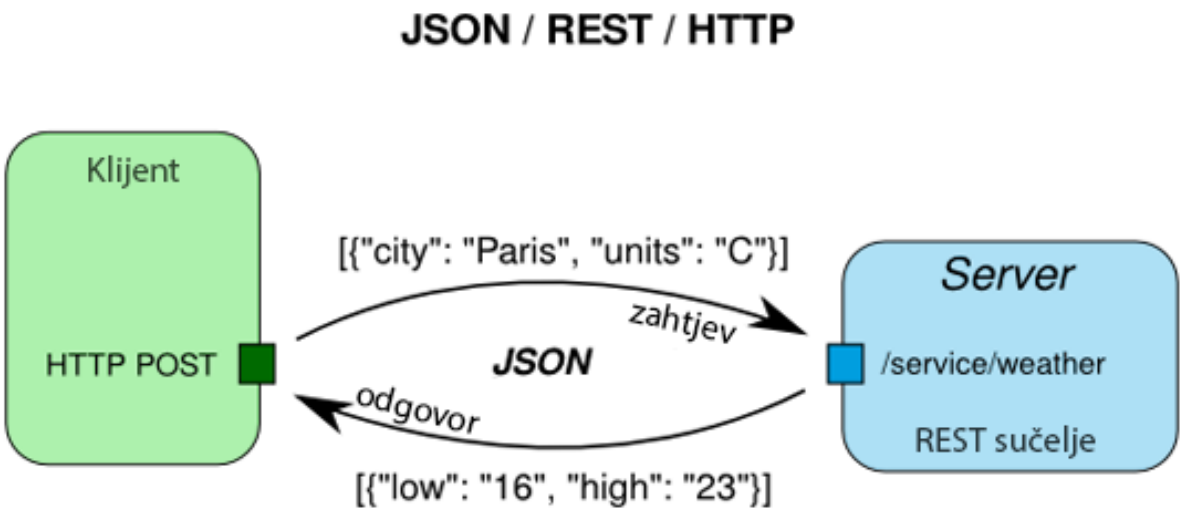
Slika 17. Shema ORM podatkovnog modela
Izvor: autor

U shemi modela “UserSchema” koriste se dvije validacijske metode koje se pozivaju prilikom serijalizacije objekte. Svaka validacijska funkcija ima dekorator “@validates()” s kojim se određuje koja varijabla se validira u toj funkciji.

Nakon deserijalizacije se koristi funkcija “wrap”, određena dekoratorom “@post_dump”, koja služi da “omotamo” dobiveni rezultati željenim tekstom.

3.7. REST

REST[15] označava programsku paradigmu za distribuirane sustave, posebno za web usluge. Nastoji stvoriti stil arhitekture za hipermedijske sustave koji bolje odražava potrebe modernog interneta. Svrha REST-a je usredotočena na komunikaciju stroj-stroj i pruža jednostavnu alternativu SOAP-u i WDSL-u. Prednost REST-a u je ta što veliki dio infrastrukture već postoji i često se koristi (serveri, HTTP klijenti, sigurnosni mehanizmi) tako da su mnoge web usluge sukladne REST-u same po sebi.



Slika 18. Prikaz jednostavnog REST zahtjeva i odgovora
Izvor: http://linkeddataorchestration.com/2014/01/28/data-modeling-for-apis-part-2-rest-and-json/

REST arhitektura propisuje pet glavnih atributa koje mora imati, ali ne opisuje način na koji bi trebali biti implementirani.

- Klijent–server arhitektura - REST mora imati sve atribute klijent-server arhitekture. Server pruža web usluga na koju se klijent može spojiti

[15] *Representational State Transfer*

- Bez stanja – svaki REST zahtjev mora sadržavati sve informacije koju su potrebne klijentu ili serveru da razumije taj zahtjev. Niti server niti klijent ne bi smjeli čuvati informacije o stanju između dvije poruke. Ovaj atribut vrlo bitan budući da dolazni REST zahtjevi mogu vrlo jednostavno biti distribuirani na različite server budući da sam zahtjev sadrži sve što je potrebno da bi se dobio valjan odgovor

- Spremanje u privremenu memoriju (Caching) – sve što se može spremiti u privremenu memoriju bi se trebalo spremiti, budući da je najbrži zahtjev onaj koji se ne mora izvršiti nego samo dohvatiti iz privremene memorije. Naravno treba paziti da zahtjev ne bude zastario

- Višeslojni sustav – individualne komponente mogu biti sakrivene iza međusloja s kojim klijent komunicira, tj, klijent se spaja na proxy i ne zna što se nalazi iza tog proxy-a. Time postižemo da su sve komponente neovisne i lako zamjenjive

- Jedinstveno sučelje - za klijenta postoji samo jedno sučelje na koje šalje zahtjev, a ostali slojevi sustava mogu biti sakriveni. Takav pristup dosta pojednostavljuje arhitekturu

Za implementaciju REST-a na aplikacijskom sloju, većinom se koriste HTTP i HTTPS protokoli koji se rašireni, jednostavni i kompatibilni sa svakim vatrozidom. Za odabir tipa operacije koju želimo izvršiti nad resursom najčešće koristimo HTTP metode GET, POST, PUT, DELETE od kojih sve osim POST moraju biti idempotentne.

- GET – koristi se za dohvaćanje resursa sa servera, ne mijenja stanje na serveru. Primjer jednog takvoj web servisa se nalazi na slici (Slika 19).

```python
@app.route("/user/")
@jwt_required
def get_current_user():
    """Get current user"""
    current_user_id = get_jwt_identity()
    user = User.query.get(current_user_id)

    result = user_schema.dump(user)

    return result.data
```

Slika 19. Primjer web servisa koji za GET zahtjev vraća podatke o trenutnom korisniku
Izvor: autor

- POST – služi za dodavanje novog resursa u kolekciju. Također može služiti za operacije koje nisu pokrivene ostalim tipovima operacija. Primjer web servisa se nalazi na slici (Slika 20)

```python
@app.route("/user/weather_stations/<int:ws_id>/weather_data", methods=["POST"])
@jwt_required
def set_user_weather_station_weather_data(ws_id):
    data = request.get_json(force=True)
    current_user_id = get_jwt_identity()
    ws = WeatherStation.query.filter_by(owner_id=current_user_id, id=ws_id).first()

    data["weather_station"] = ws

    weather_data = WeatherData(**data)
    weather_data.save()

    cache.delete_memoized(WeatherData.get_latest_data_for_weather_station_id,ws.id, weather_data_latest_schema)
    cache.delete("view/{}".format(url_for("get_latest_weather_data")))

    result = weather_data_schema.dump(weather_data)
    return result.data
```

Slika 20. Primjer web servisa koji za POST zahtjev dodaje novi meteorološki zapis u bazu podataka
Izvor: autor

- PUT – najčešće se koristi za promjenu atributa nekog resursa. Primjer web servisa se nalazi na slici (Slika 21)

```python
@app.route("/user/", methods=["PUT"])
@jwt_required
def edit_current_user():
    """Edit current user"""
    data = request.get_json(force=True)

    allowed_changes = ["first_name", "last_name", "email"]
    for k in data.keys():
        if k not in allowed_changes:
            data.pop(k, None)

    current_user_id = get_jwt_identity()
    user = User.query.get(current_user_id)
    user = user_schema.load(data, instance=user, partial=True)
    user = user.data
    user.save()

    result = user_schema.dump(user)

    return result.data
```

Slika 21. Primjer web servisa koji za PUT zahtjev vrši promjenu atributa
Izvor: autor

- DELETE – služi za brisanje resursa. Primjer web servisa se nalazi na slici (Slika 22)

```python
@app.route("/user/weather_stations/<int:ws_id>/", methods=["DELETE"])
@jwt_required
def delete_user_weather_station(ws_id):
    current_user_id = get_jwt_identity()
    ws = WeatherStation.query.filter_by(owner_id=current_user_id, id=ws_id).first()
    ws.delete()

    return jsonify({"status":"deleted"})
```

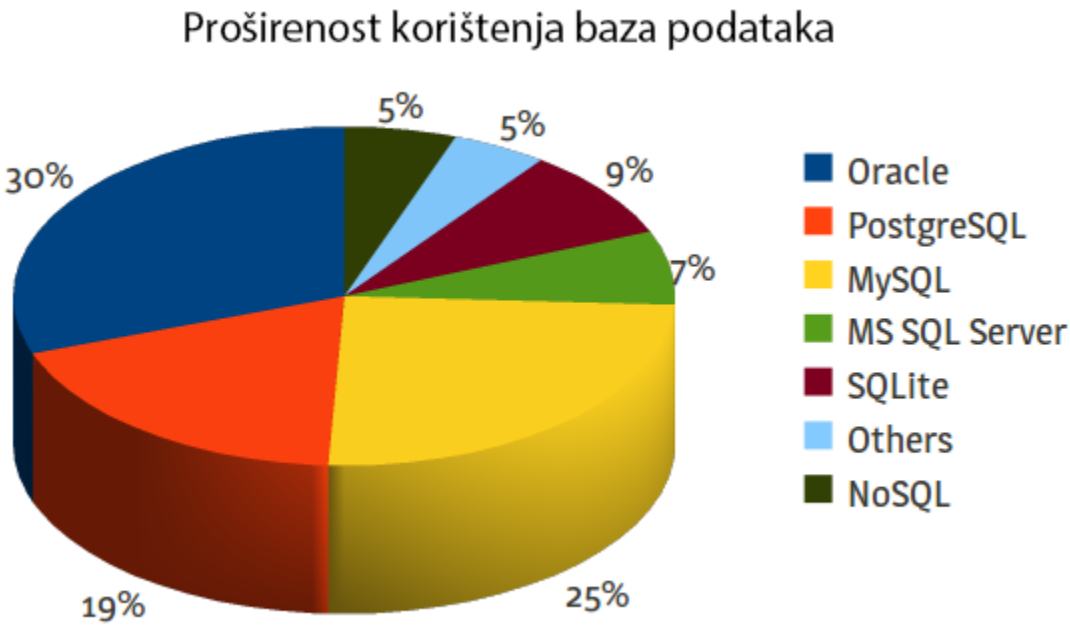
Slika 22. Primjer web servisa koji za DELETE zahtjev briše taj resurs
Izvor: autor

Uz pomoći primjera sa slika, vidi se jednostavnost implementacije web servisa koristeći Python i Flask zbog čega se ta kombinacija viđa u dosta web aplikacija.

3.8. PostgreSQL baza podataka

PostgreSQL (Postgres) je besplatni objektno relacijski sustav za upravljanje bazama podataka. S razvojem je započeto u 1980-im dok je prva verzija izdana 1997. godine kao program otvorenog koda. Postgres je većinom sukladan SQL standardu i većina funkcija je dostupna i ponaša kao što je definirano u standardu dok također postoje funkcije koje su specifične samo za Postgres. Postgres je i u potpunosti sukladan s ACID principom. Sam Postgres može biti proširen od korisnika na razne načine, kao što su dodavanje novih:

- Tipova podataka
- Funkcija
- Operatora
- Agregatnih funkcija
- Proceduralnih jezika
- Metoda indeksa



Slika 23. Proširenost korištenja različith baza podataka
Izvor: https://launchschool.com/books/sql_first_edition/read/preparations

U praktičnom dijelu diplomskog rada koristimo bazu podatka “diplomski” koja je stvorena naredbom “createdb diplomski” u terminalu, a čiji SQL kod možemo vidjeti na slici (Slika 24).

```
-- Database: diplomski

-- DROP DATABASE diplomski;

CREATE DATABASE diplomski
    WITH
    OWNER = nino
    ENCODING = 'UTF8'
    LC_COLLATE = 'en_US.UTF-8'
    LC_CTYPE = 'en_US.UTF-8'
    TABLESPACE = pg_default
    CONNECTION LIMIT = -1;
```

Slika 24. Kreiranje baze u PostgreSQL-u
Izvor: autor

U nastavku na slikama Slika 25, Slika 27 i Slika 26 možemo vidjeti SQL kod za kreiranje ostalih najbitnijih tablica u praktičnom dijelu rada.

```
-- Table: public.users

-- DROP TABLE public.users;

CREATE TABLE public.users
(
    id integer NOT NULL DEFAULT nextval('users_id_seq'::regclass),
    date_created timestamp without time zone,
    date_modified timestamp without time zone,
    first_name character varying(255) COLLATE pg_catalog."default",
    last_name character varying(255) COLLATE pg_catalog."default",
    username character varying(255) COLLATE pg_catalog."default" NOT NULL,
    email character varying(255) COLLATE pg_catalog."default" NOT NULL,
    email_confirmed_at timestamp without time zone,
    password_hash character varying(255) COLLATE pg_catalog."default" NOT NULL,
    active boolean,
    CONSTRAINT users_pkey PRIMARY KEY (id),
    CONSTRAINT users_email_key UNIQUE (email)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.users
    OWNER to nino;
```

Slika 25. Kreiranje tablice "users"
Izvor: autor

Na slici (Slika 25) se vidi kreiranje tablice “users” koja se koristi za pohranu podataka o registriranim korisnicima web aplikacije.

```
-- Table: public.weather_stations

-- DROP TABLE public.weather_stations;

CREATE TABLE public.weather_stations
(
    id integer NOT NULL DEFAULT nextval('weather_stations_id_seq'::regclass),
    date_created timestamp without time zone,
    date_modified timestamp without time zone,
    name character varying(255) COLLATE pg_catalog."default" NOT NULL,
    coordinates geometry(Point,4326) NOT NULL,
    owner_id integer NOT NULL,
    above_sea_level double precision,
    active boolean,
    CONSTRAINT weather_stations_pkey PRIMARY KEY (id),
    CONSTRAINT weather_stations_name_key UNIQUE (name),
    CONSTRAINT weather_stations_owner_id_fkey FOREIGN KEY (owner_id)
        REFERENCES public.users (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.weather_stations
    OWNER to nino;

-- Index: idx_weather_stations_coordinates

-- DROP INDEX public.idx_weather_stations_coordinates;

CREATE INDEX idx_weather_stations_coordinates
    ON public.weather_stations USING gist
    (coordinates)
    TABLESPACE pg_default;
```

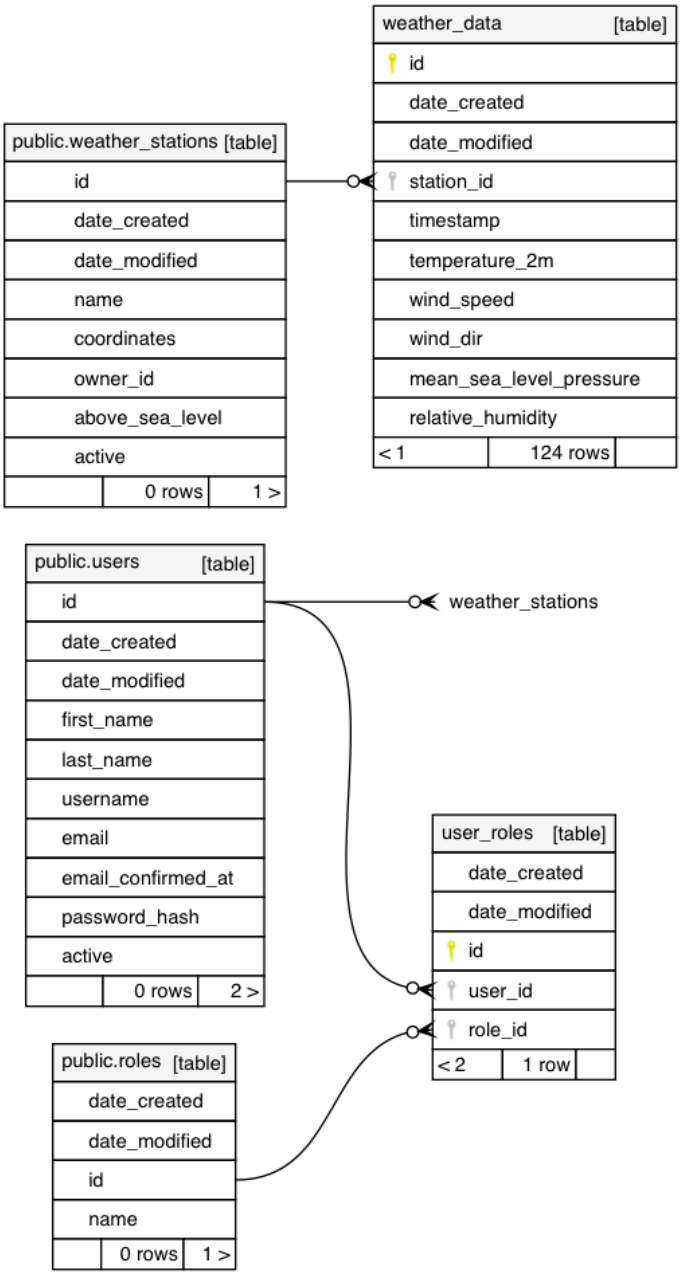
Slika 26. Kreiranje tablice "weather_stations"
Izvor: autor

Kreiranje tablice “weather_data” se nalazi na slici (Slika 26). Tablica će se koristiti za spremanje podataka o meteorološkim postajama kao što su ime postaje, vlasnik, koordinate i sl.

```
-- Table: public.weather_data

-- DROP TABLE public.weather_data;

CREATE TABLE public.weather_data
(
    id integer NOT NULL DEFAULT nextval('weather_data_id_seq'::regclass),
    date_created timestamp without time zone,
    date_modified timestamp without time zone,
    station_id integer NOT NULL,
    "timestamp" timestamp without time zone,
    temperature_2m double precision,
    wind_speed double precision,
    wind_dir double precision,
    mean_sea_level_pressure double precision,
    relative_humidity double precision,
    CONSTRAINT weather_data_pkey PRIMARY KEY (id),
    CONSTRAINT weather_data_station_id_fkey FOREIGN KEY (station_id)
        REFERENCES public.weather_stations (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.weather_data
    OWNER to nino;
```

Slika 27. Kreiranje tablice "weather_data"
Izvor: autor

Slika (Slika 27) pokazuje kod za kreiranje tablice “weather_data”. Tablica se koristi za spremanje meteoroloških zapisa s postaja te se u nju spremaju podaci poput temperature, brzine vjetra, relativne vlažnosti te ostali meteorološki podaci.



Slika 28. Shema baze podataka "diplomski"
Izvor: autor

Na slici (Slika 28) je vidljiva kompletna shema baze podataka “diplomski”.

3.8.1. PostGIS

Iako PostgreSQL već podržava vrste geometrija, to nije bilo dovoljno za cjelovito pohranjivanje i analizu prostornih podataka zbog čega je 2000. godine razvijen PostGIS. PostGIS je proširenje objektno-relacijske baze podataka Postgres koje uključuje geografske/geometrijske objekte i funkcije. Implementira specifikacije jednostavnog pristupa značajkama OGC-a [16]. Podržava sljedeće vrste geometrija[17]:

- Point
- Linestring
- Polygon
- MultiPoint
- MultiLineStrings
- MultiPolygon
- GeometryCollections

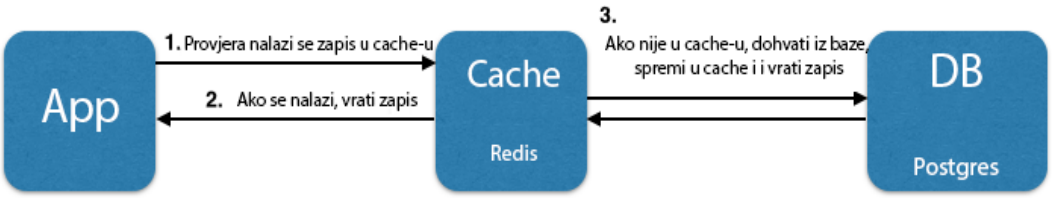
Uz vrste geometrije, također podržava prostorne funkcije kao što su izračunavanje udaljenosti, razna preklapanje geometrija, analiza rastera i vektorskih podataka i sl.

Pristup PostGIS-u se obavlja istim alatima i sučeljima kao i Postgres bazi podataka dok veliki broj GIS programa podržava PostGIS nativno.

3.9. Redis - caching

Redis je najkorištenija baza podataka koja se nalazi unutar memorije s ključ-vrijednost strukturom podataka. Takva jednostavna struktura baze podataka je manje prikladna za složene strukture podataka, ali velika prednost Redisa je ta što je dosta brži od relacijskih baza podataka. Zbog brzinu pristupa i pohranu, često se koristi sa spremanje i dohvaćanje podataka iz memorije, tj. za *caching*.

[16] Open Geospatial Consortium
[17] Roy Thomas Fielding, Architectural Styles and the Design of Network-based Software Architectures, UNIVERSITY OF CALIFORNIA, IRVINE, 2000



Slika 29. Primjer rada sustava s bazom podataka i cache-om
Izvor: https://dzone.com/articles/enabling-caching-in-mongodb-database-with-redis-us

Caching radi tako da prilikom dohvaćanje nekog zapisa prvo provjerimo nalazi li se taj zapis u *cache-u,* ako se nalazi onda zapis vraćamo aplikaciji. Ako zapis ne postoji onda se dohvaća iz baze podataka, sprema u *cache* te se vraća aplikaciji (Slika 29).

```
@app.route("/weather_stations/weather_data/latest", methods=["GET"])
@cache.cached(timeout=5*60)
def get_latest_weather_data():
    response_format = request.args.get("format", None)
    weather_stations = WeatherStation.get_all()
    data = []
    for ws in weather_stations:
        wd = WeatherData.get_latest_data_for_weather_station_id(ws.id, weather_data_latest_schema)
        if not wd:
            continue

        ws_data = weather_station_latest_schema.dump(ws).data
        ws_data.update({"weather_data":wd})
        data.append(ws_data)

    if response_format == "geojson":
        data = converters.dict_to_geo_json(data)

    else:
        data = {"weather_stations":data}

    return data
```

Slika 30. Primjer caching na view-u
Izvor: autor

Na slici (Slika 30) imamo *view* koji služi za dohvaćanje zadnjih meteoroloških zapisa. Na njemu koristimo decorator "@cache.cached(timeout=5*60)" koji će

provjeriti postoji li vrijednost ovog view-a već u predmemoriji (*cache-u*) i da li je ključ za taj view noviji od pet minute, ako je sve to zadovoljeno onda će se korisniku vratiti vrijednost iz predmemorije. U slučaju da ključ nije u predmemoriji ili da je vrijednost starija od pet minute, onda će se izvršiti funkcija, spremiti ključ i vrijednost u predmemoriji te će se korisniku vratiti ta vrijednost

```
@staticmethod
@cache.memoize(60*60)
def get_latest_data_for_weather_station_id(id, schema=None):
    wd = WeatherData.query.filter_by(station_id=id).order_by("timestamp desc").first()
    if schema:
        return schema.dump(wd).data
    else:
        return wd
```

Slika 31. Primjer spremanja rezultata funkcije u memoriju
Izvor: autor

Spremanje u predmemoriju se može napraviti za svaku funkciju u web aplikaciji kao na slici (Slika 31).

```
@app.route("/user/weather_stations/<int:ws_id>/weather_data", methods=["POST"])
@jwt_required
def set_user_weather_station_weather_data(ws_id):
    data = request.get_json(force=True)
    current_user_id = get_jwt_identity()
    ws = WeatherStation.query.filter_by(owner_id=current_user_id, id=ws_id).first()

    data["weather_station"] = ws

    weather_data = WeatherData(**data)
    weather_data.save()

    cache.delete_memoized(WeatherData.get_latest_data_for_weather_station_id,ws.id, weather_data_latest_schema)
    cache.delete("view/{}".format(url_for("get_latest_weather_data")))

    result = weather_data_schema.dump(weather_data)
    return result.data
```

Slika 32. Primjer zaprimanja zapisa te brisanja njegove vrijednosti iz predmemorije
Izvor: autor

Ponekad je potrebno u web aplikaciji izbrisati vrijednost u predmemoriju iako još nije

istekla. U praktičnom dijelu diplomskog rada takva funkcionalnost se koristi prilikom primanja novog meteorološkog zapisa. Nakon što web aplikacija zaprimi i spremi zapis, vrijednost predmemorije se za taj ključ briše (Slika 32).

3.10. JWT autentifikacija

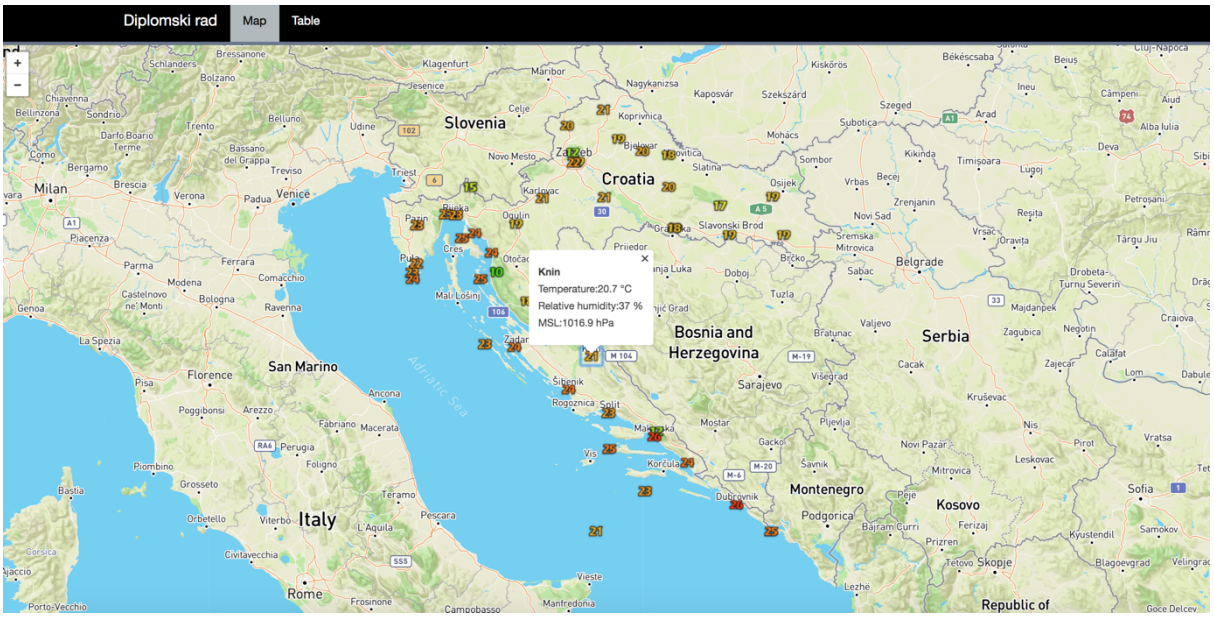
JSON Web Token (JWT) je standardizirani autentifikacijski token baziran na JSON-u. JWT omogućuje razmjenu povjerljivih zahtjeva te se obično koristi za razmjenu identiteta korisnika. JWT se sastoji od tri dijela:

- Zaglavlje – JSON element koji opisuje koja metoda šifriranja se koristi
- Korisna nosivost – JSON element koji opisuje entitet, npr. Korisnika
- Potpis – služi za verifikaciju poruke

Token možemo prenijet HTTP protokolom unutar samog url-a kao GET parametar ili u zaglavlju HTTP zahtjeva.

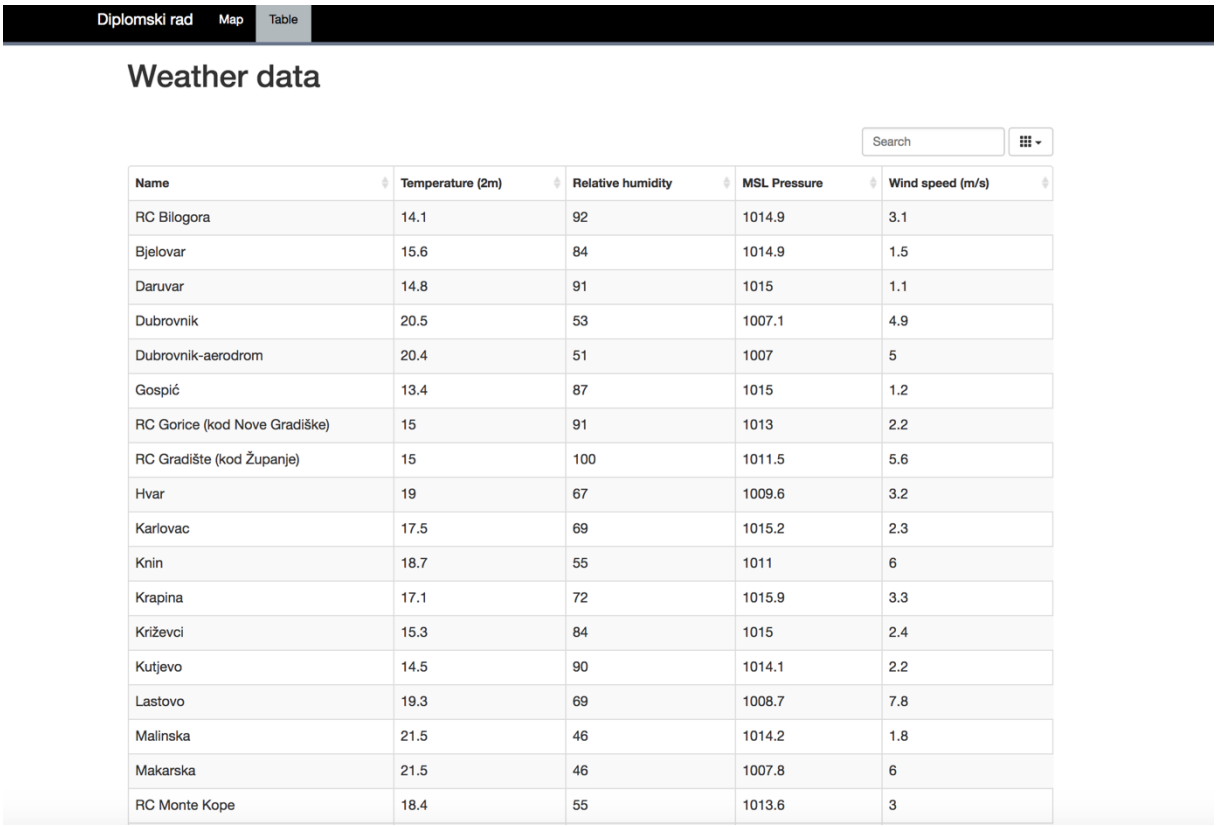
4. WEB APLIKACIJA

Koristeći sve navedene tehnologije u teorijskom dijelu rada, napravljena je jednostavna web aplikacija koja služi za prikaz i arhiviranje meteoroloških zapisa. Veći dio sustava se nalazi u *backend* dijelu aplikacije dok je *frontend* dio napravljen kao pokazatelj nekih funkcionalnosti sustava. Na početku same web aplikacije se nalazi karta s prikazom meteoroloških postaja u Hrvatskoj s njihovim aktualnim podacima. Ikone prikazuju trenutnu temperaturu dok se pritiskom na svaku pojedinu postaju dobiva detaljniji prikaz meteorološkog stanja. Cijeli *view* je napravljen tako da se koristi Jinja2 predložak u kojeg se ubacuju podaci o meteorološkim postajama. Prikaz ove karte se sprema u privremenu memoriju s rokom trajanja od pet minuta ili do prvog novog meteorološkog zapisa, ovisno što je kraće



Slika 33. Prikaz karte u web aplikaciji
Izvor: autor

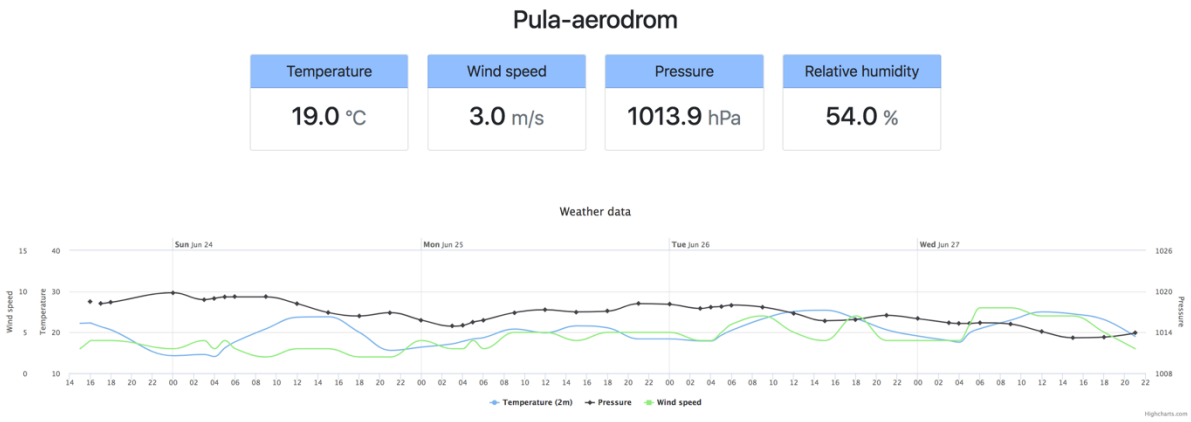
U web aplikaciji također postoji tablični prikaz lokacija koji radi na sličan način kao prikaz karte. Podatke u tablici je moguće pretraživati te je moguće mijenjati poredak ovisno o vrijednosti nekog parametra.



Name	Temperature (2m)	Relative humidity	MSL Pressure	Wind speed (m/s)
RC Bilogora	14.1	92	1014.9	3.1
Bjelovar	15.6	84	1014.9	1.5
Daruvar	14.8	91	1015	1.1
Dubrovnik	20.5	53	1007.1	4.9
Dubrovnik-aerodrom	20.4	51	1007	5
Gospić	13.4	87	1015	1.2
RC Gorice (kod Nove Gradiške)	15	91	1013	2.2
RC Gradište (kod Županje)	15	100	1011.5	5.6
Hvar	19	67	1009.6	3.2
Karlovac	17.5	69	1015.2	2.3
Knin	18.7	55	1011	6
Krapina	17.1	72	1015.9	3.3
Križevci	15.3	84	1015	2.4
Kutjevo	14.5	90	1014.1	2.2
Lastovo	19.3	69	1008.7	7.8
Malinska	21.5	46	1014.2	1.8
Makarska	21.5	46	1007.8	6
RC Monte Kope	18.4	55	1013.6	3

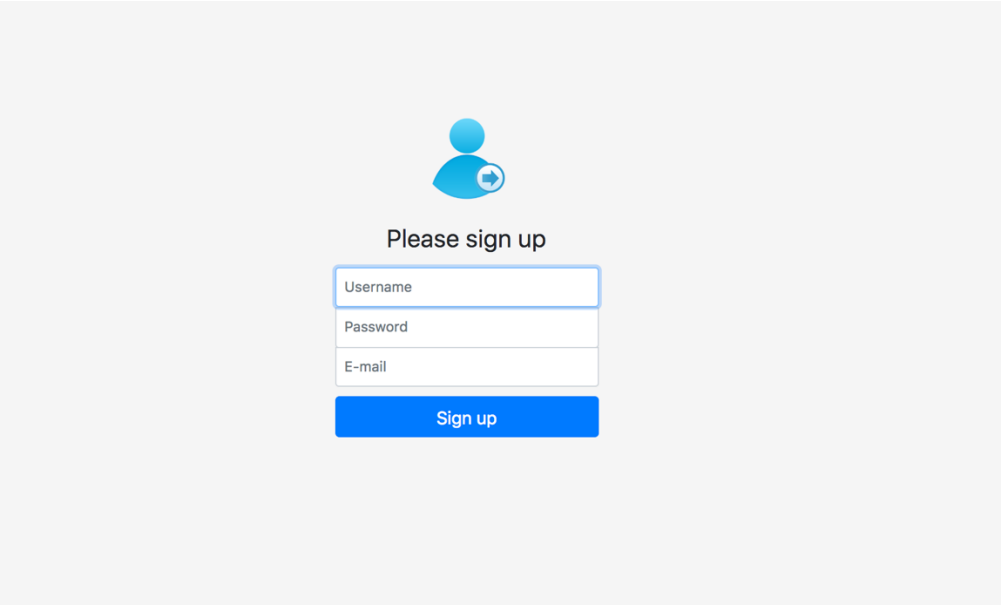
Slika 34. Tablični prikaz meteoroloških zapisa
Izvor: autor

Pritiskom na jednu od lokacija u tablici, otvara se pojedinačni prikaz podataka s te meteorološke stanice.



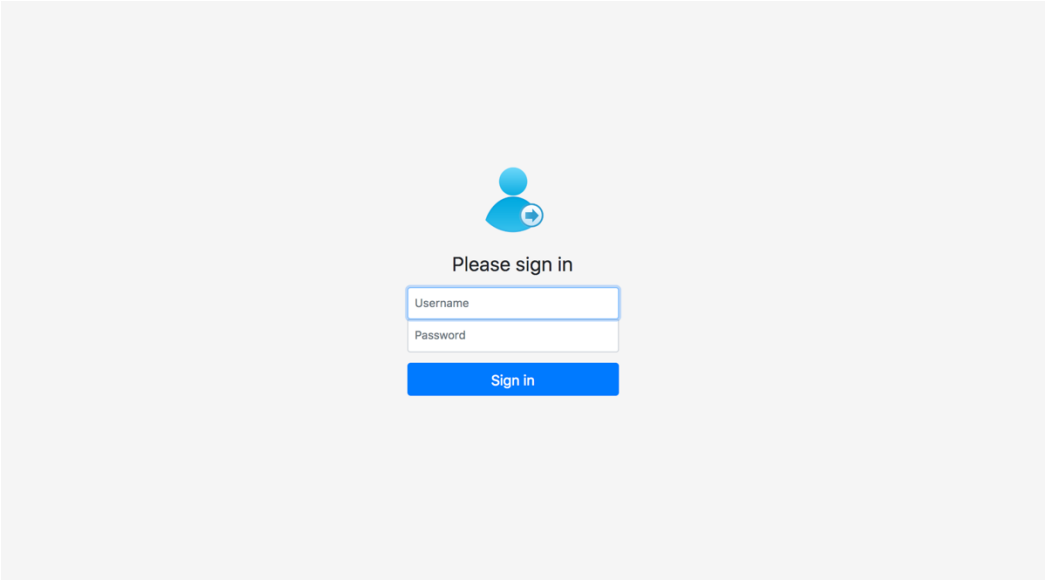
Slika 35. Pojedinačni prikaz podataka s meteorološke stanice
Izvor: autor

Da bi korisnik mogao slati podatke sa svoje meteorološke postaje, mora se prvo registrirati, nakon čega dobiva JWT token s kojim može kreirati meteorološke stanice te slati podatke.



Slika 36. Forma za registraciju
Izvor: autor

Također, postoji ista takva forma koja služi za prijavu korisnika nakon čega korisnik opet dobiva svoj JWT token.



Slika 37. Forma za prijavu korisnika
Izvor: autor

Primjer funkcija za dohvaćanje meteoroloških postanja, kreiranje novih te kreiranje novog meteorološkog zapisa preko REST API-a možemo vidjeti na slici (Slika 38)

```
BASE_URL = "http://127.0.0.1:5000"

def get_weather_stations_from_db():
    url = "{base_url}/user/weather_stations/".format(base_url=BASE_URL)
    headers = {"authorization": "Bearer {}".format(jwt)}
    response = requests.get(url, headers=headers)
    data = response.json

    ws_list = []
    for ws in data["weather_stations"]:
        ws_list.append((ws["name"],ws["id"]))

    return dict(ws_list)

def import_weather_station_in_db(name, location):
    url = "{base_url}/user/weather_stations/".format(base_url=BASE_URL)
    headers = {"authorization": "Bearer {}".format(jwt)}
    data = {"name":name, "location":location}

    response = requests.post(url, headers=headers, json=data)

    return response.json()["weather_stations"]

def import_weather_data(ws_id, data):
    url = "{base_url}/user/weather_stations/{ws_id}/weather_data".format(base_url=BASE_URL, ws_id=ws_id)
    headers = {"authorization": "Bearer {}".format(jwt)}
    requests.post(url, headers=headers, json=data)
```

Slika 38. Primjer REST API upita
Izvor: autor

Funkcije sa slike (Slika 38) se mogu iskoristiti za slanje podataka s meteorološke postaje tako da se postaja spoji na kompjuter koristeći serijski port te se svakih nekoliko minuta pročitaju aktualni podaci nakon čega se uz gore navedene funkcije pomoću HTTP zahtjeva mogu poslati u web aplikaciju.

5. ZAKLJUČAK

U teorijskom djelu diplomskog rada se obrađivao razvoj i metodologije izrade web aplikacija te su obrađene tehnologije korištene u praktičnom djelu. Rezultat diplomskog rada je bio napraviti jednostavnu web aplikaciju za prikaz i spremanje meteoroloških podataka. Prilikom izrade aplikacije korišten je iterativni razvoj s kojim se prvo napravila baza aplikacije te su se na nju iterativno razvijale nove mogućnosti. Vizualni dio aplikacije (frontend) je napravljen koristeći HTML, Javascript i CSS tehnologije dok je za backend korišten Python kao glavni programski jezik, PostgreSQL za bazu podataka te Redis za rad s predmemorijom. Zbog brzine i kvalitete razvoja aplikacije, korišten je Flask framework pomoću kojeg se s lakoćom mogla napraviti web aplikacija koja zadovoljava REST programsku paradigmu. Koristeći REST API vrlo je jednostavno spojiti *frontend* i *backend* te se omogućilo jednostavno slanje meteoroloških zapisa u web aplikaciju. Za autentifikaciju korisnika, korišten je JWT kojeg se sve češće viđa kod razvoja web aplikacija.

LITERATURA

1. Oxford Living Dictionaries, dostupno na https://en.oxforddictionaries.com/definition/us/web_app , pristupljeno 06.2018.
2. TechTerms, dostupno na: https://techterms.com/definition/web_application, pristupljeno 06.2018.
3. LifeWire, dostupno na: https://www.lifewire.com/what-is-a-web-application-3486637 , pristupljeno 06.2018.
4. Mobidev, dostupno na: https://mobidev.biz/blog/3_types_of_web_application_architecture, pristupljeno 06.2018.
5. Nitin Uikey, Ugrasen Suman, A Lifecycle Model for Web-based Application Development: Incorporating Agile and Plan-driven Methodology, 2015.
6. R.Manger, Softversko inženjerstvo (skripta), Prirodoslovno matematički fakultet u Zagrebu, 2005.,
7. Agilna metodologija, dostupno na: https://dusanmilosevic.com/agilna-metodologija/, pristupljeno 06.2018.
8. Melsata.blog, dostupno na: https://melsatar.blog/2012/03/15/software-development-life-cycle-models-and-methodologies/, pristupljeno 06.2018.
9. Python Documentation, dostupno na: https://docs.python.org/3/license.html, pristupljeno 06.2018.
10. WikiPedia, dostupno na: https://hr.wikipedia.org/wiki/Python_(programski_jezik), pristupljeno 06.2016
11. WikiPedia, dostupno na: https://en.wikipedia.org/wiki/Python_(programming_language), pristupljeno 06.2018.
12. Communications of the ACM, dostupno na: https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext, pristupljeno 06.2018.
13. StackOverflow, dostupno na: https://stackoverflow.blog/2017/09/06/incredible-growth-python/, pristupljeno 06.2018.

14. Shuup, dostupno na: https://www.shuup.com/blog/25-of-the-most-popular-python-and-django-websites/, pristupljeno 06.2018.
15. Quora, dostupno na: https://www.quora.com/Which-Python-web-framework-was-YouTube-built-with-when-they-started-off, pristupljeno 06.2018.
16. ExpandedDramblings, dostupno na: https://expandedramblings.com/index.php/dropbox-statistics/, pristupljeno 06.2018.
17. Instagram-engineering, dostupno na: https://instagram-engineering.com/web-service-efficiency-at-instagram-with-python-4976d078e366, pristupljeno 06.2018.
18. Miguel Grinberg, Flask Web Development, O'Reilly, 2014., dostupno na: http://aksitha.com/Data%20Base/SQL%20Database/Essential%20SQLAlchemy,%202nd%20Edition.pdf, pristupljeno 06.2018.
19. Django adapters documentation, dostupno na: https://media.readthedocs.org/pdf/django-adapters/latest/django-adapters.pdf, pristupljeno 06.2018.
20. Roy Thomas Fielding, Architectural Styles and the Design of Network-based Software Architectures, UNIVERSITY OF CALIFORNIA, IRVINE, 2000
21. Davor Lozić, dr.sc. Alen Šimec, Pametna komunikacija na Internetu preko REST protokola, Tehničko veleučilište, dostupno na: https://bib.irb.hr/datoteka/713376.2014_MIPRO_Pametna_komunikacija_na_Internetu_preko_REST_protokola.pdf, pristupljeno 06.2018.
22. PostGIS documentation, dostupno na: http://postgis.net/documentation/, pristupljeno 06.2018.
23. Redis documentation, dostuno na: https://redis.io/documentation, pristupljeno 06.2018.
24. Introduction to JSON Web Tokens, dostupno na: https://jwt.io/introduction/, pristupljeno 06.2018.

POPIS SLIKA

Slika 1. Prikaz server-side arhitekture Izvor: https://mobidev.biz/blog/3_types_of_web_application_architecture 4

Slika 2. Prikaz AJAX arhitekture Izvor: https://mobidev.biz/blog/3_types_of_web_application_architecture 5

Slika 3. Prikaz Service-orinted single page web applications arhitektura Izvor: https://mobidev.biz/blog/3_types_of_web_application_architecture 6

Slika 4. Model vodopada Izvor: http://ttl.masfak.ni.ac.rs/SUK/Softversko%20inzenjestvo.pdf..... 8

Slika 5. Inkrementalni razvoj aplikacije Izvor: http://ttl.masfak.ni.ac.rs/SUK/Softversko%20inzenjestvo.pdf..... 10

Slika 6. Agilna SCRUM metoda Izvor: https://www.info-novitas.hr/o-nama/metodologije-rada/scrum-procesni-framework/ 12

Slika 7. Postotak pitanja postavljenih o pojedinom programskom jeziku Izvor: https://stackoverflow.com 16

Slika 8. Postotak pitanja postavljenih o pojedinom programskom jeziku u zemljama s visokim nacionalnim dohotkom Izvor: https://stackoverflow.com 17

Slika 9. Primjer jednostavne Flask web aplikacije Izvor: autor 19

Slika 10. Prikaz rada Flask aplikacije Izvor: https://gertjanvanhethofblog.wordpress.com/2016/02/10/exploring-flask-on-raspberry-pi/ 20

Slika 11. Prikaz više virtualnih okruženja na jednom stroju izvor: https://learnbatta.com/blog/how-to-install-python-virtualenv-19/ 21

Slika 12. Primjer template-a koristeći Jinja 2 Izvor: autor..... 22

Slika 13. Evaluacija varijabli u Python-u te renderiraranje predloška Izvor: autor..... 22

Slika 14. ORM podatkovni model User Izvor: autor 24

Slika 15. ORM podatkovni model WeatherStation Izvor: autor 26

Slika 16. ORM podatkovni model WeatherData Izvor: autor 27

Slika 17. Shema ORM podatkovnog modela Izvor: autor 28

Slika 18. Prikaz jednostavnog REST zahtjeva i odgovora Izvor: http://linkeddataorchestration.com/2014/01/28/data-modeling-for-apis-part-2-rest-and-json/ 29

Slika 19. Primjer web servisa koji za GET zahtjev vraća podatke o trenutnom korisniku Izvor: autor 31

Slika 20. Primjer web servisa koji za POST zahtjev dodaje novi meteorološki zapis u bazu podataka Izvor: autor 31

Slika 21. Primjer web servisa koji za PUT zahtjev vrši promjenu atributa Izvor: autor 32

Slika 22. Primjer web servisa koji za DELETE zahtjev briše taj resurs Izvor: autor .. 32

Slika 23. Proširenost korištenja različith baza podataka Izvor: https://launchschool.com/books/sql_first_edition/read/preparations 33

Slika 24. Kreiranje baze u PostgreSQL-u Izvor: autor 34

Slika 25. Kreiranje tablice "users" Izvor: autor 35

Slika 26. Kreiranje tablice "weather_stations" Izvor: autor 36

Slika 27. Kreiranje tablice "weather_data" Izvor: autor 37

Slika 28. Shema baze podataka "diplomski" Izvor: autor 38

Slika 29. Primjer rada sustava s bazom podataka i cache-om Izvor: https://dzone.com/articles/enabling-caching-in-mongodb-database-with-redis-us 40

Slika 30. Primjer caching na view-u Izvor: autor 40

Slika 31. Primjer spremanja rezultata funkcije u memoriju Izvor: autor 41

Slika 32. Primjer zaprimanja zapisa te brisanja njegove vrijednosti iz predmemorije Izvor: autor 41

Slika 33. Prikaz karte u web aplikaciji Izvor: autor 43

Slika 34. Tablični prikaz meteoroloških zapisa Izvor: autor 44

Slika 35. Pojedinačni prikaz podataka s meteorološke stanice Izvor: autor 45

Slika 36. Forma za registraciju Izvor: autor 45

Slika 37. Forma za prijavu korisnika Izvor: autor 46

Slika 38. Primjer REST API upita Izvor: autor 47

Sažetak

Cilj ovog diplomskog rada je bio razviti jednostavnu web aplikaciju za prikupljanje i analiziranje meteoroloških podataka. U radu su opisane tehnologije i koncepti koji su korišteni za izradu aplikaciju te su opisane metode razvoja softvera, tj. web aplikacija. Tehnologije korištene u radu su HTML, Javascript i CSS za razvoj *frontend-a* te Python, PostgreSQL i Redis za razvoj *backend-a*. Programsko rješenje je napisano da zadovoljava REST programsku paradigmu s kojom se olakšava komunikacija s ostalim dijelovima sustava te su riješeni problemi spremanja i dohvaćanja podataka u predmemoriju te autentifikacija korisnika koristeći najmodernija rješenja.

Summary

The aim of this master thesis was to develop a simple web application for collecting and analyzing meteorological data. The paper describes the technologies and concepts used to create the application and describes methods of software development, ie web applications. The technologies used in the paper are HTML, Javascript and CSS for frontend development and Python, PostgreSQL and Redis for backend development. The software solution was written to comply with REST programming paradigm, which ease the communication with other parts of the system. This paper also addresses the issues of saving and retrieving data in the cache and authenticating users using state-of-the-art solutions.