

Sveučilište u Rijeci – Odjel za informatiku
Jednopredmetni preddiplomski studij informatike

Ivan Modrić

Izrada web aplikacije korištenjem Flaska

Završni rad

Mentor: dr. sc. Vedran Miletić

Rijeka, 23. rujna 2021.

Sažetak

Ovaj završni rad donosi pregled razvoja programskih jezika i alata za razvoj web aplikacija. Potom se prikazuje arhitektura cloud aplikacija temeljena na tri osnovna modela te prema metodologiji 12 faktora. Zatim slijedi detaljan opis izrade vlastite Flask web aplikacije u sinergiji s AWS cloud servisima poput virtualnog poslužitelja EC2, Lambde, DynamoDB baze podataka, SNS servisa i CloudWatch događaja.

Ključne riječi

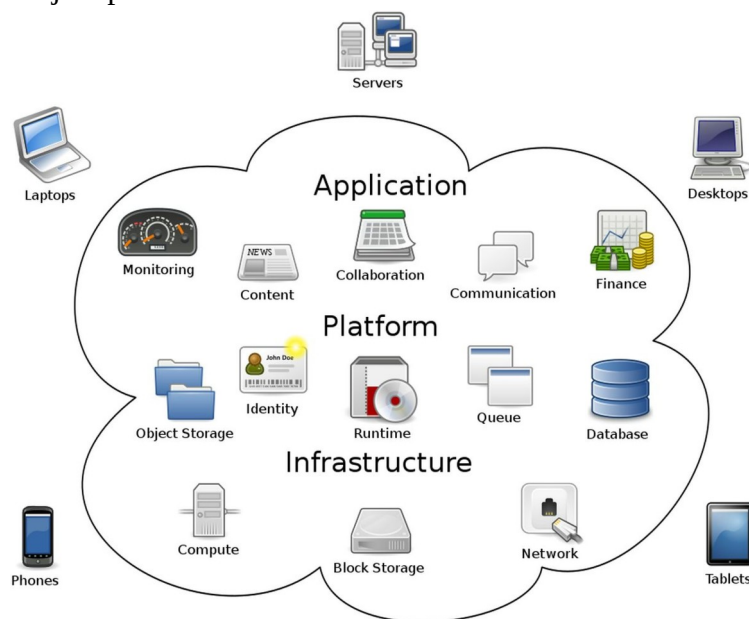
flask, cloud, web aplikacija, aws, lambda, dynamodb, python, sns, cloudwatch event, trigger, ec2

Sadržaj

1. Uvod.....	1
2. Evolucija dinamičkih web tehnologija.....	2
2.1. Common Gateway Interface (CGI).....	2
2.1.1. Modul CGI.pm.....	3
2.2. Hypertext Preprocessor (PHP).....	4
2.3. Ruby.....	5
2.3.1. Ruby on Rails (RoR).....	5
2.4. Django.....	7
2.5. Node.js.....	8
2.5.1. Express.js.....	9
2.6. Flask.....	9
3. Arhitektura aplikacije u oblaku.....	11
3.1. Modeli usluga u oblaku.....	11
3.2. Metodologija 12 faktora.....	12
4. Primjer cloud web aplikacije u Flasku.....	18
4.1. Motivacija.....	18
4.2. Arhitektura i korištena tehnologija.....	19
4.2.1. Kreiranje rječnika.....	19
4.2.2. Algoritam za pretraživanje i izvlačenje podataka.....	20
4.2.3. Uvođenje threadinga.....	22
4.2.4. Ispis.....	23
4.2.5. Izrada Flask web aplikacije.....	23
4.2.6. Dizajn sučelja web aplikacije.....	25
4.2.7. Uvođenje Amazon web servisa.....	26
4.2.8. Izrada logike za slanje obavijesti.....	29
5. Zaključak.....	31
6. Literatura.....	33

1. Uvod

Razvojem informacijske i komunikacijske tehnologije te mnogobrojnih usluga koje tržište nudi došlo je do sve veće želje, a i potrebe za poboljšanjem i unaprjeđenjem tih usluga. Upravo ta snažna želja i potreba za modernizacijom dovela je do pojave i daljnjeg razvoja računarstva u oblaku (eng. *cloud computing*), odnosno clouda. Nova cloud tehnologija u potpunosti je promijenila odnose među korisnicima, podacima i resursima. Primjerice, fizička infrastruktura za pohranu i čuvanje podataka sve je manje potrebna zbog lakoće pristupa cloud servisima na kojima se jednostavno i jeftino mogu pohraniti velike količine podataka. Cloud karakteriziraju niski troškovi korištenja te fleksibilnost i pouzdanost korištenja. Brigu oko hardvera, softvera i njegovih nadogradnji najčešće preuzimaju administratori, a ne sami korisnici. U slučaju da dođe do pada jednog servera, podaci koji su se nalazili na njemu, gotovo će odmah biti dostupni na drugom serveru bez štete po korisnika. Podaci su pokretni i lako dostupni u svakom trenutku. Korištenje clouda tvrtkama omogućuje bolji fokus na tržište, korisnike i na sam esencijalni posao pa se ne trebaju brinuti o infrastrukturi, sigurnosti, održavanju te investiranju u novu infrastrukturu i zaposlenike koji bi se bavili time. Tako se ostvaruju značajne financijska uštede, ali i uštede na vremenu kojeg se potom može iskoristiti za nove projekte, ujedno tako pridonoseći razvoju tržišne inovativnosti. Za korištenje clouda osnovni je uvjet pristup internetu, što s jedne strane u današnje vrijeme ne bi trebalo predstavljati problem, dok s druge strane ujedno predstavlja nedostatak u slučaju slabe internetska veze ili ako je u prekidu.



Slika 1. Elementi računalstva u oblaku
Izvor: wikipedia.org

2. Evolucija dinamičkih web tehnologija

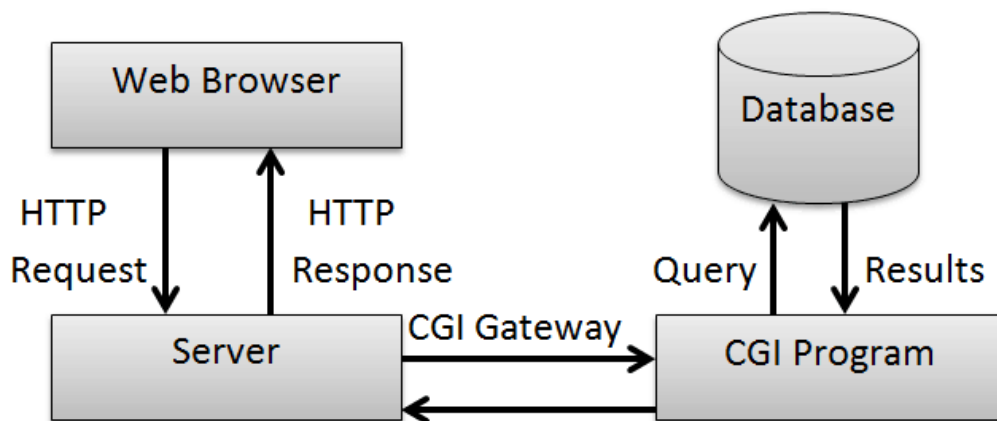
U proteklom desetljeću, a posebice zadnjih nekoliko godina vidljiv je značajan trend zamjene statičkih web stranica dinamičkim. Dinamičkim ih čini promjenjiv sadržaj koji prikazuju, dok statičke stranice ostaju iste sve dok ih netko ne promijeni ručno. Pisane su najčešće samo u jezicima HTML i CSS te su time dizajnerski veoma jednostavne. Kod statičkih stranice ne koriste se baze podataka. One jednostavno preko web preglednika šalju HTTP zahtjeve na web poslužitelj te od njega dobivaju HTTP odgovore. Kod dinamičkih je priča složenija, a time i uzbudljivija. Dinamičke web stranice puno su interaktivnije s korisnicima, ali time i mnogo složenije za izradu.

2.1. Common Gateway Interface (CGI)

Gledajući povijesni razvitak dinamičkih stranica u samom početku može se uočiti korištenje CGI-a (engl. *Common Gateway Interface*). Početak CGI-a je 1993. godina kad je tim iz Nacionalnog centra za superračunalne aplikacije (eng. *National center for supercomputing applications*, skraćeno NCSA) napisao specifikaciju za pozivanje izvršnih datoteka naredbenog retka na mailing listi www-talk, a drugi web programeri su je prihvatili te je postala standard za web poslužitelje. Formalnu definiciju CGI dobio je tek 1997. godine kad je radna skupina, koju je predvodio Ken Coar, formalno definirala NCSA definiciju CGI-ja.

CGI je sučelje koje je web serverima omogućavalo pokretanja programa kao odgovora na HTTP zahtjeve koji stižu s web preglednika. CGI program, koji je najčešće bio pisan u programskim jezicima Perl ili C, zatim je čitao informacije koje web preglednik šalje preko web poslužitelja te vraćao generirani odgovor nakon čega se program prekida. Glavni nedostatak je bio u tome što programi ovog tipa nisu bili proširivi. Uvijek se morao pisati novi program ispočetka jer se stari programi, koji su već radili na serverima, nisu mogli nadograđivati.

Postoji i opcija korištenja DBI (eng. *Database Interface*) ekstenzije za Perl koja služi za oblikovanje SQL upita i za čitanje podataka sadržanih u bazi podataka. Kad se podaci vrate onda se formatiraju i šalju klijentu. U ovoj varijanti se preko CGI programa pristupa bazi podataka.



Slika 2. Prikaz rada CGI sučelja
Izvor: networkencyclopedia.com

2.1.1. Modul CGI.pm

CGI.pm objektno je orijentirani Perl modul koji služi za stvaranje i parsiranje CGI obrazaca. Distribuirao se s jezgrom Perla od verzije 5.004, a danas više nije dio jezgre. Korištenje CGI.pm modula moglo je znatno olakšati pisanje programskog koda. Veoma se jednostavno koristio polučujući bitne efekte svoje uporabe o čemu svjedoči i njegova ogromna popularnost među Perl programerima svih razina.

Primjerom ispod prikazano je ispunjavanje i slanje obrasca za datum rođendana. Kao i kod svakog Perl modula, prvo treba pozvati modul naredbom *use*. Zatim se poziva konstruktor *new()* koji kreira novi CGI objekt naziva *\$query*. Nakon toga, korištenjem *param()* metode dobit ćemo vrijednost objekta *\$bday*. CGI.pm samostalno odlučuje hoće li CGI program biti pozvan s metodom GET ili POST te samostalno vrši dekodiranje URL-a. Za generiranje izlaza koristi se *header()* metoda koja govori da je tip sadržaja header te *p()* metoda koja generira oznake za tagove paragrafa *<p>*.

```
#!/usr/bin/perl -w
# cgi script with CGI.pm

use CGI;

$query = CGI::new();
$bday = $query->param("birthday");
print $query->header();
print $query->p("Your birthday is $bday.");
```

Kad bi se ovaj program pisao bez upotrebe *cgi.pm* modula, skripta bi ručno morala prevoditi kodirani URL, vjerojatno koristeći niz regularnih izraza te tek ga tada dodijeliti varijabli. Time

zaključujemo da cgi.pm može znatno olakšati i skratiti proceduru. Štoviše, ovo je samo mali dio dobrobiti ovog modula. Postoje četiri osnovne kategorije objektno-orijentiranih metoda u CGI.pm modulu: metode za rukovanje (eng. handling), stvaranje obrazaca, dohvaćanje varijabli te stvaranje HTML oznaka.

2.2. Hypertext Preprocessor (PHP)

PHP je zasigurno jedan od najpoznatijih programskih jezika koji se koristi za razvoj web aplikacija. Sintaksa PHP-a bazira se na programskim jezicima C, Perl i Java. Programerima omogućuje brzo razvijanje programa proceduralnim ili objektno-orijentiranim načinom. PHP je, između ostalog, postao popularan i zbog jednostavnog dodavanja biblioteka u okruženje.

PHP je nestao 1994. godine. Njegov začetnik je Rasmus Lerdorf, a kasnije se u njegov daljnji razvitak uključilo mnoštvo programera. Isprva je skraćenica označavala *Personal Home Page*, a danas označava *PHP: Hypertext Preprocessor* što ujedno označava i glavnu funkciju jezika, a to je generiranje HTML-a na temelju PHP naredbi. U početku je PHP bio samo pomoćni faktor Rasmusu Lerdorfu prilikom izrade vlastite web stranice tako što je dotadašnje programe pisane u Perlu zamijenio s CGI programima pisanim u programskom jeziku C. Prvo i druga verzija PHP-a nisu bile od velikog značaja, dok je treća verzija, koja je puštena u pogon 1998. godine, izazvala ogromno zanimanje. PHP danas koriste popularne web stranice poput Facebooka ili Wikipedije, a trenutno je u uporabi osma verzija PHP-a.

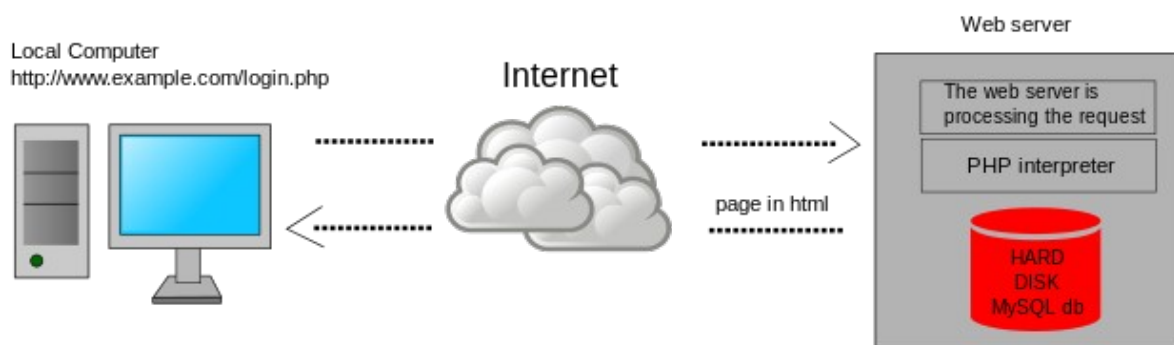
PHP je jezik otvorenog koda i svaka njegova verzija kreirana je koristeći unose od strane programera. Tako jezik može bolje i kvalitetnije napredovati jer sami korisnici mogu upravljati njime. Posebnost programa otvorenog koda je omogućavanje svima zainteresiranim da neposredno sudjeluju u razvijanju programa, a povrhu toga su besplatni za korištenje. Za razliku od programa otvorenog koda, postoje i programi zatvorenog koda kao što je to primjerice Microsoftov C# kojeg kreira i uređuje sama kompanija.

Novije verzije PHP-a često se mogu preuzeti u paketima otvorenog koda kao što su WAMP, LAMP, MAMP ili XAMPP. Navedeni paketi olakšavaju procese instalacije jer se više proizvoda instalira odjednom. Spomenuti paketi sadrže PHP za specifičan operacijski sustav (npr. Linux), web server (npr. Apache) i bazu podataka (npr. MySQL).

Kad pristupamo web stranici web preglednik šalje zahtjev poslužitelju koji mu vraća datoteku u HTML-u. Zatim preglednik korisniku tu datoteku prikazuje grafički. U počecima su te datoteke bile na poslužiteljima, a takav oblik smještaja naziva se dvoslojnom arhitekturom. Danas se koristi

troslojna arhitektura u kojoj datoteke nisu smještene direktno na poslužitelju, već u bazi podataka s kojom poslužitelj komunicira. Prvi sloj je klijentski, a čine ga preglednik povezan na internetsku mrežu. Srednji sloj je web poslužitelj koji izvršava skriptne jezike i datoteke, a treći sloj je onaj u kojem se nalazi baza podataka i sustav za upravljanje njome.

Kako bi poslužitelj znao da treba izvršavati PHP kod, datoteci je potrebno nadodati ekstenziju „.php”. Poslužitelj prvo šalje PHP stranicu u PHP procesor koji je obrađuje tj. prevodi. Dok se kod obrađuje procesor traži od baze podataka eventualno obrađivanje SQL upita. Sustav za upravljanje bazom podataka potom vraća rezultate SQL upita u PHP procesor. PHP procesor onda formatira podatke za ispis na poslužitelj. Tek tada poslužitelj vraća na web preglednik podatke koje mu je vratio PHP procesor zajedno s HTML-om, eventualno CSS-om, JavaScriptom itd. Web preglednik na kraju prevodi HTML i eventualno JavaScript i CSS i finalni rezultat se prikazuje korisniku.



Slika 3. Prikaz procesuiranja PHP-a na web poslužitelju
Izvor: wikipedia.org

2.3. Ruby

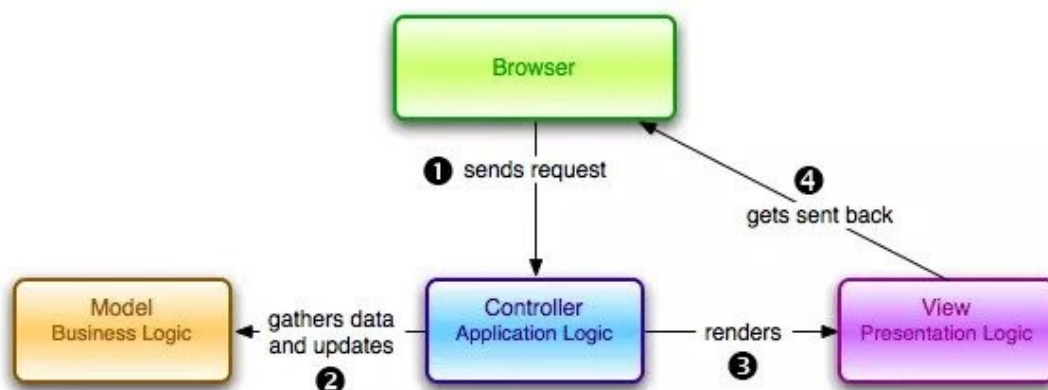
Ruby je potpuno objektno-orijentirani programski jezik opće namjene. Kao objektno-orijentirani jezik dobar je za izradu velikih projekata jer svaki programer može dobiti dio zadatka pa svaki može raditi na samo jednoj klasi. Pisan je otvorenim kodom. Ruby je 1995. godine razvio Yukihiro Matsumoto. Sintaksa je vrlo jednostavna i laka za čitanje što će se posebno svidjeti početnicima, no mana mu je brzina izvođenja. Popularnost mu raste od 2005. godine. Postoji nekoliko Ruby programskih okvira (eng. *framework*), a najpopularniji su Sinatra i Rails. Sinatra je kreiran 2007. godine. Dobar je za početnike i koristi se za brzu izradu web aplikacija uz minimalan napor pa ga se još zove i mikro okvirom (eng. *microframework*).

2.3.1. Ruby on Rails (RoR)

Ruby on Rails je web programski okvir koji se koristi za razvoj web aplikacija pomoću jezika Ruby

i programskog okvira Rails koji je zapravo skup biblioteka koje su tu da pomognu pri razvoju web aplikacije. Aplikaciju izrađenu na ovaj način će gotovo svaki Rails programer u jako malo vremena razumjeti te biti u stanju nadograditi. Zbog toga je suradnja među timovima programera olakšana i izbjegavaju se česte greške koje se događaju u slučaju kad se web aplikacije rade pomoću standardnih web tehnologija i programskih jezika. Rails programski okvir nastao je 2004. godine kad je PHP bio glavni na sceni i kad su se web aplikacije izrađivale načinima koje su HTML, CSS i PHP dozvoljavali, a Ruby je odskakujući od tadašnjih standarda privukao značajnu pozornost javnosti.

Ruby on Rails temelji se na MVC, odnosno Model-View-Controller arhitekturi. Čine ju modeli za upravljanje podacima i poslovnom logikom, pogledi za upravljanje objektima grafičkog korisničkog sučelja (GUI) i kontrolori za upravljanje korisničkim sučeljem i aplikacijom. U praksi to znači da web preglednik šalje zahtjev za stranicu kontroloru na web poslužitelju, zatim kontrolor preuzima podatke koje treba iz modela kako bi odgovorio na zahtjev nakon čega kontrolor daje preuzete podatke pogledu i na posljepku se generira pogled i šalje natrag klijentu u web preglednik.



Slika 4. Prikaz MVC arhitekture
Izvor: sitepoint.com

Korištenje Ruby on Railsa ima mnoge prednosti. Veoma je lako naučiti osnove korištenja te postoji mnoštvo alata koji mogu ubrzati i pojednostaviti postupke. Također postoji mnoštvo biblioteka koje je zajednica napravila i koje se mogu koristiti u vlastitom programu. Neke pomažu u otklanjanju pogrešaka, neke optimiziraju kod, a neke su tu da bi pomogle kod testiranja. Naravno, postoje i određene mane. Gledajući osnovne značajke programskog okvira, RoR je gotovo nenadmašiv. Međutim, s obzirom na to da postoji mnoštvo zadanih, već postavljenih objekata, nije preostalo puno prostora za kreativnost, što RoR ne čini naročito fleksibilnim. Kontinuirani razvoj mu je u jednu ruku prednost, ali u drugu i mana. Početnicima se teško prilagoditi na stalne promjene u

frameworku, alatima i bibliotekama pa je potrebno biti dio zajednice kako bi se moglo biti u toku sa svim promjenama.

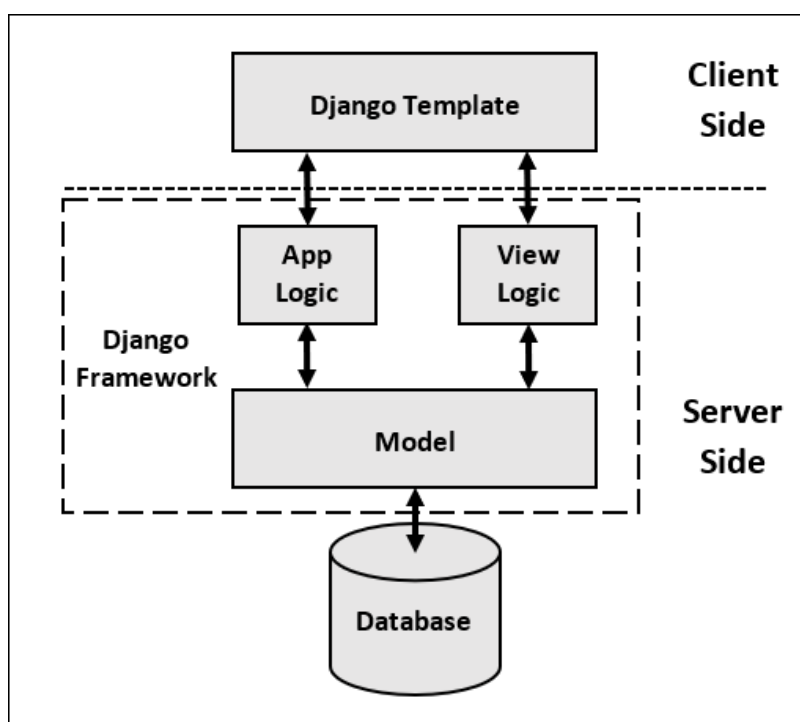
2.4. Django

Ono što je Ruby on Rails za Ruby, to je Django za Python. Django je napisan u programskom jeziku Python, otvorenog je koda pa je stoga besplatan za korištenje. Njegov početak je 2003. godina kada dvojac iz Lawrence Journal Worlda, Simon Willison i Adrian Holovaty, započinju korištenje Pythona kod izrade aplikacija. Django je u javnosti tek od srpnja 2005. godine kad je okvir bio već dovoljno razvijen, a naziv je dobio prema jazz gitaristu 20. stoljeća Django Reinhardt.

Django je razvijan u novinarskom okruženju zbog toga što su novinari zahtijevali da se stranice mogu izraditi veoma brzo i jednostavno. Django je stoga posebice prilagodljiv za sadržajne stranice koje daju dinamičke informacije i koje su povezive s bazom podataka. Naravno, Django je itekako koristan i za ostale vrste dinamički generiranih web stranica. Django se kao okvir učestalo poboljšava i usavršava zbog toga što je njegov kod otvoren, a njegove stranice lako se održavaju, dobro se ponašaju pod opterećenjem te se mogu napraviti u kratkom roku.

Iako se Django temelji na već spomenutoj MVC (Model-View-Controller) arhitekturi, ipak postoji određena razlika s obzirom na to da neke arhitekture koje koriste taj obrazac mogu varirati u pojedinim obilježjima. Prema striktnoj definiciji MVC arhitekture prvi dio je model, sloj koji obrađuje logiku povezanu s podacima. Na primjer, može dohvatiti, promijeniti i spremi podatke u bazu podataka. Drugi dio je pogled (eng. *view*), a može se nazvati i prezentacijskim slojem jer je odgovoran za prikupljanje podataka od modela ili korisnika i prezentiranje istih. U web aplikaciji, sve što je prikazano u web pregledniku spada pod pogled. Treći dio je kontrolor koji je posrednik u komunikaciji između modela i pogleda. Primjerice, kontrolor može na temelju zahtjeva prikupiti podatke iz baze podataka pomoću modela i poslati ih korisniku putem pogleda. Django se temelji na MVT (Model-View-Template) arhitekturi koji je potekla iz MVC arhitekture. Glavna razlika između MVC i MVT arhitekture je što je za MVC potrebno napisati specifičan i određen kod za kontrolor, dok kod MVT arhitekture postoji programski okvir koji preuzima brigu kontrolora. Uzmimo za primjer da imamo zadatak u jednoj tablici naziva „knjige” prikazati popis knjiga koje imamo u određenoj knjižnici. Kod MVC arhitekture morali bi napisati kod koji dohvaća popis knjiga iz baze podataka i napraviti prezentacijski sloj (HTML, CSS itd.), te ga poslati na URL-u korisniku. Kad bi bila u pitanju MVT arhitektura, ne bi trebalo pisati nikakav kod tog tipa, već bi

svim tim aktivnostima upravljao sam okvir. Jedini zadatak bio bi reći okviru koje podatke bi uopće trebao prikazati korisniku, u ovom slučaju tablicu „knjige”. Okvir bi tada sam stvorio prikaz i poslao ga korisniku. Druga razlika očituje se u sljedećem. Kad bi korisnik kod MVC arhitekture izvršio neku radnju ili uputio zahtjev, pozvao bi se kontrolor. On bi tada rekao modelu da izvrši promjene na pogledu ili bi vratio pogled na temelju modela, stoga zaključujemo da kontrolor i model upravljaju pogledom. Kod MVT arhitekture nakon što bi korisnik izvršio neku radnju ili uputio zahtjev, odgovarajući bi pogled izvršavao upit na modelu i prikupljao rezultate od njega. Pogled bi zatim ispisivao rezultat u predlošku (eng. *template*) te rezultat slao korisniku. Za razliku od MVC-a, kod MVT-a pogled nije povezan s modelom i to ga čini slabo povezanim, ali jednostavnim za rađanje nekakvih izmjena.



Slika 5. Prikaz Django arhitekture
Izvor:.djangobook.com

2.5. Node.js

JavaScript je programski jezik nastao 1995. godine, a pretežno se koristi u web preglednicima uz HTML i CSS. S obzirom na to da ipak nije striktno ograničen na uporabu u web preglednicima, jedno od rješenja za korištenje JavaScripta izvan web preglednika je Node.js kojeg je razvio Ryan Dahl 2009. godine. Dvije godine kasnije postao je projekt otvorenog koda. Node.js je JavaScriptov okvir koji je zadovoljio potrebu da se JavaScript izvodi na bilo kojem računalu izvan web preglednika. Baziran je na Google Chrome JavaScript engineu V8 kojem je prvotna svrha bila

korištenje u web pregledniku, no kasnije je prešao na otvoreni kod i tako postao dobra podloga Node.js-u. Zbog V8 enginea, Node.js odlikuje velika brzina izvršavanja programskog koda. Glavni cilj postojanja Node.js-a je pružanje sigurnosti i lakoće pri izradi aplikacija visokih performansi u JavaScriptu. Sve API-ji (eng. *Application Programming Interface*) Node.js biblioteke su asinkroni, odnosno ne blokiraju. To konkretno znači da poslužitelji zasnovani na Node.js-u nikada ne čekaju da API vrati podatke, već poslužitelj odmah prelazi na novi API nakon što ga se pozove, a mehanizam obavijesti pomaže poslužitelju da dobije odgovor iz prethodnog API poziva. Node.js koristi model s jednom niti s ponavljanjem događaja (eng. *event looping*) i ovaj model može pružiti uslugu za puno veći broj zahtjeva od tradicionalnih poslužitelja poput Apache web servera. Mehanizam događaja pomaže poslužitelju šaljući odgovore tako da se poslužitelj ne blokira te ga tako čini visoko skalabilnim (može procesirati podatke brže i efikasnije) za razliku od tradicionalnih poslužitelja koji stvaraju ograničene niti za obradu zahtjeva. Node.js aplikacije nikada ne spremaju podatke u međuspremnik, već ih jednostavno odmah ispisuju u komadima. Izdvojio bih i da Node.js posjeduje MIT licencu koja dozvoljava ponovno korištenje softvera dokle god sve kopije softvera sadrže cjeloviti tekst i uvjete licence.

2.5.1. Express.js

Postoje mnogi Node.js dodaci kao što su Express.js, Compound.js, derby.js, sails.js, partial.js i ostali. Osvrnuo bih se na Express.js dodatak koji je postao gotovo standardni poslužiteljski okvir za Node.js. Express.js dolazi kao back-end komponenta kod popularnih paketa za razvoj web aplikacija kao što je primjerice MEAN, kojeg još čine MongoDB (baza podataka), AngularJS (front-end komponenta) i Node.js. Uspoređujući ovaj paket s ranije spomenutim LAMP-om kod PHP-a, sadržaj MEAN paketa sadrži i prezentacijski sloj, dok ne sadrži sloj operacijskog sustava.

Express.js najkorišteniji je Node.js-ov alat koji programskom kodu omogućuje konstantno i beskonačno izvođenje. Express.js omogućuje postavljanje posredničkih programa (eng. *middleware*) za odgovaranje na HTTP zahtjeve, definira tablicu usmjeravanja koja se koristi za izvođenje različitih radnji temeljem HTTP metoda i URL-ova te omogućuje dinamičko generiranje HTML stranica temeljem prenošenja argumenata u predloške.

2.6. Flask

Flask je mali web okvir koji se temelji na programskom jeziku Python. Stvorio ga je Armin Ronacher 2004. godine kao prvoaprilsku šalu. Neovisno o tome, brzo je stekao popularnost u

zajednici otvorenog koda te opstao kao popularan projekt s puno sljedbenika, a opstoji i danas. Definiran je kao mikro okvir jer ne zahtijeva posebne alate ili biblioteke. Mikro ne znači da cijelu aplikaciju mora činiti jedna Python datoteka (iako može), niti da Flask nema dovoljno funkcionalnosti. To zapravo znači da je Flaskova svrha zadržavanje jezgre jednostavnom, ali lako proširivom. Flask uglavnom neće samostalno donositi odluke umjesto programera, ali će se lako prilagoditi željama i potrebama programera. Flask svojim dizajnom nudi osnovne usluge okvira, ali ga je lako proširiti s mnogim nadogradnjama tako da korisnik sam može odabrati što želi koristiti kod izrade aplikacije pa nema bespotrebnog gomilanja koda. To omogućuje sažet, jasan i razumljiv programski kod. Flask možda je mikro, ali je u svakom trenutku spreman zadovoljiti potrebe većine.

Prema dogovoru, predlošci (eng. *templates*) i statičke datoteke (eng. *static files*) pohranjuju se u poddirektorije naziva „static” i „templates” unutar izvornog stabla aplikacije. Predlošci, odnosno templatei datoteke su koje sadrže statičke podatke, kao i rezervirana mjesta za dinamičke podatke. Za njihovo renderiranje Flask koristi engine Jinja2.

Minimalna Flask aplikacija oblika je:

```
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

if __name__ == '__main__':
    app.run()
```

Prvo se uvozi klasa Flask, a primjer ove klase je WSGI (eng. *Server Gateway Interface*) aplikacija. Koristeći klasu Flask koja se uvozi stvara se instanca klase. Prvi argument je naziv modula ili paketa. U većini slučajeva je to `__name__`, a potreban je kako bi Flask znao gdje tražiti resurse poput templatea ili static datoteka. Nakon toga dodaje se `route()` dekorator koji Flasku govori koji URL pokreće funkciju koja vraća poruku koja se želi prikazati u web pregledniku. U ovom slučaju to je jedan HTML paragraf koji ispisuje *Hello, World!* Za kraj, aplikacije se može lokalno pokrenuti.

3. Arhitektura aplikacije u oblaku

Cloud je tehnologija koja se u današnje vrijeme tiče svih poslovnih organizacija, od najmanjih do najvećih. Stoga je potrebno više no ikad znati razlike između različitih modela usluga u oblaku, neovisno radilo se o implementaciji samo jedne aplikacije ili cijele infrastrukture.

Svaki od spomenutih modela ima određene prednosti pa je od ključne važnosti znati i razumjeti sva tri modela.

3.1. Modeli usluga u oblaku

Postoje tri glavna modela usluga u oblaku:

1. Software as a Service (SaaS), odnosno Softver kao usluga

Najčešće je korišten model za poslovanje u oblaku. SaaS aplikacije ne zahtijevaju nikakva preuzimanja ni instaliranja od strane korisnika pa se najčešće pokreću direktno putem nekog od web preglednika što ih čini izrazito fleksibilnima. To znači da ne postoje nikakva dodatna ulaganja u poslužitelje ili licence programa, a troškovi se svode na minimum u odnosu na tradicionalno držanje datoteka na poslužitelju. Kod SaaS modela tehničkim problemima i poteškoćama bave se oni koji isporučuju SaaS, a ne izravno oni koji ga koriste. Također korisnici ne moraju gubiti vrijeme na instalaciju te nadogradnje softvera.

2. Platform as a Service (PaaS), odnosno Platforma kao usluga

Platforma kao usluga u principu je SaaS model koji kao uslugu omogućuje razvojno okruženje. Korisnik tako može definirati, napraviti i održavati aplikacije koje će se pokretati na infrastrukturi davatelja usluge, no svim poslužiteljima, pohranom i umrežavanjem upravlja upravo taj dobavljač usluge. Takav princip web programerima pruža slobodu i fokus na izgradnju aplikacija, a ne ostale popratne uobičajene procedure.

3. Infrastructure as a Service (IaaS), odnosno Infrastruktura kao usluga

Infrastruktura kao usluga pruža korisniku mogućnost korištenja virtualne računalne infrastrukture. Funkcionira na principu samoposluživanja za pristupanje i nadzor računala, umrežavanje, pohranu i ostalo. Omogućuje se kupnja resursa prema stvarnim potrebama korisnika te se virtualno isporučuje sva potrebna infrastruktura poput poslužitelja, operativnog sustava, mreže i sustava za pohranu. IaaS upravljanje svojim uslugama nudi preko nadzorne ploče gdje se korisniku omogućuje potpuna

kontrola kupljenje infrastrukture. Ipak, korisnici IaaS-a ne upravljaju izravno poslužiteljima, hard diskovima, virtualizacijom, mrežom i pohranom i sličnim, već to rade davatelji usluga. Može se reći da je ovo najfleksibilniji model računalstva u oblaku. Kad god korisnik nije najsigurniji u zahtjeve novih aplikacija, zbog skalabilnosti i fleksibilnosti, IaaS je najbolje rješenje.



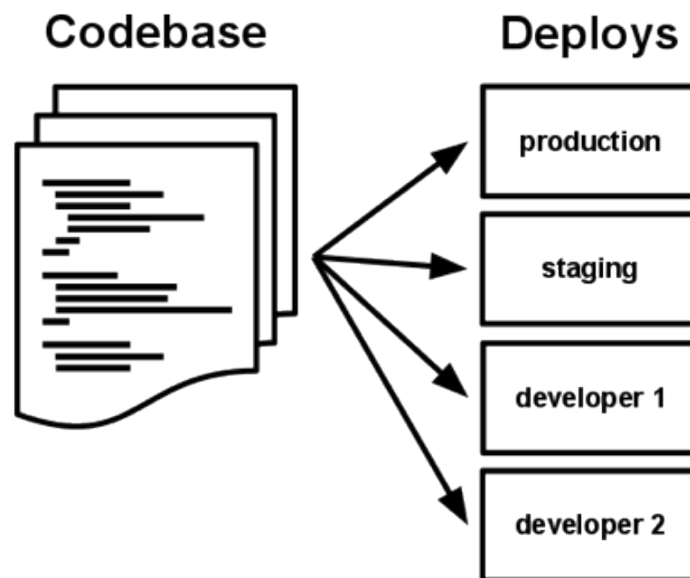
Slika 6. Prikaz modela usluga u oblaku
Izvor: edge.siriuscom.com

3.2. Metodologija 12 faktora

U današnje vrijeme softver se najčešće isporučuje kao usluga, odnosno web aplikacija. Prema Metodologiji 12 faktora (eng. *Twelve-Factor App*), koju je 2011. godine predstavio Adam Wiggins, postoji 12 faktora koji su mjerilo izrade uspješne web aplikacije. Wigginsova metodologija omogućuje stvaranje koda koji se može pouzdano objaviti, brzo skalirati tj. nadograditi te održavati na pouzdan i predvidiv način.

1. Baza kodova (eng. *codebase*) – jedna baza kodova, a mnogo implementacija aplikacije

Načelo baze kodova nalaže postojanje samo jedne baze kodove po aplikaciji uz više mogućih implementacija. Kad bi postojalo više baza kodova to bi onda bio distribuirani sustav, a ne aplikacija. Više aplikacija koje dijele isti kod krše ovo pravilo te se preporučuje isti kod uvrstiti u biblioteku koja se može kasnije uključiti i u ostale aplikacije.



Slika 7. Prikaz korelacije baze kodova i raznih implementacija
Izvor: 12factor.net

2. Ovisnosti (eng. *dependencies*) – eksplicitno deklariranje i izoliranje ovisnosti

Drugo načelo ovisnosti oslanja se na izričitu deklaraciju postojanja svih ovisnosti putem manifesta deklaracije ovisnosti. Deklaracija ovisnosti potrebna je jer omogućuje jednostavnije postavljanje za programere koji tek počinju raditi na aplikaciji. Potrebno je također koristiti i alat za izolaciju ovisnosti koji osigurava da nikakve implicitne ovisnosti neće procuriti iz sustava. Primjer takvih alata su Maven, Npm, Bundler, NuGet itd. Eksplicitna, odnosno potpuna ovisnost jednoliko bi se trebala primjenjivati i kod faze razvoja i kod faze proizvodnje.

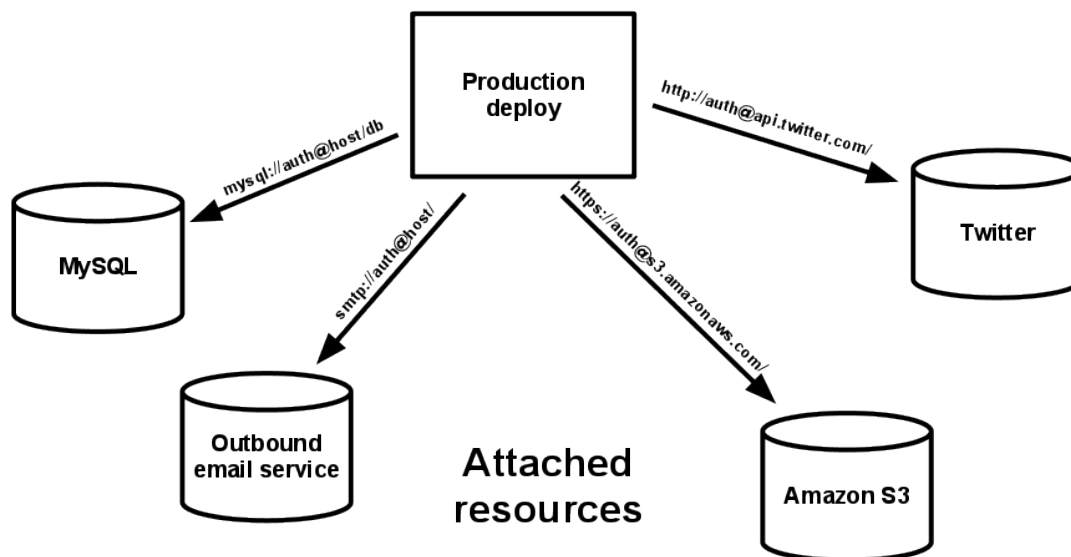
3. Konfiguracija (eng. *config*) – spremanje konfiguracije u okruženje (eng. *environment*)

Ako aplikacija ima spremljenu svoju konfiguraciju kao konstantnu u kodu to krši ovo načelo. Načelom konfiguracije nalaže se pohranjivanje konfiguracije u varijable okruženja zato što se konfiguracija znatno razlikuje kroz razne implementacije. Neovisnom konfiguracijom aplikaciji se poboljšava fleksibilnost. Primjerice, prednost držanja konfiguracije odvojeno od logike aplikacije očituje se u tome što se može imati jedan skup konfiguracije za implementaciju namijenjenu okruženju za testiranje, a drugi skup za implementaciju namijenjenu okruženju za proizvodnju.

4. Pomoćne usluge (eng. *backing services*) – tretiranje pomoćnih usluga kao odvojenih resursa

Pomoćna usluga svaka je usluga koju aplikacija koristi putem mreže tijekom svojeg

normalnog rada. To mogu biti usluge spremanja podataka poput MySQL-a ili DynamoDB-a, nekih sustava za razmjenu poruka, nekih SMTP usluga za slanje emailova i slično. Ako se poštuje četvrto načelo pomoćnih usluga ako dođe do promijene podataka o lokaciji ili povezivanju na pomoćnu uslugu ne treba se mijenjati ništa u kodu. Takvi detalji trebaju biti dostupni u konfiguraciji. Primjerice, ako jedna pomoćna usluga ne radi kako treba, samo se promijeni u cloud okruženju bez intervencija u kodu.



Slika 8. Prikaz pomoćnih usluga povezanih s aplikacijom
Izvor: 12factor.net

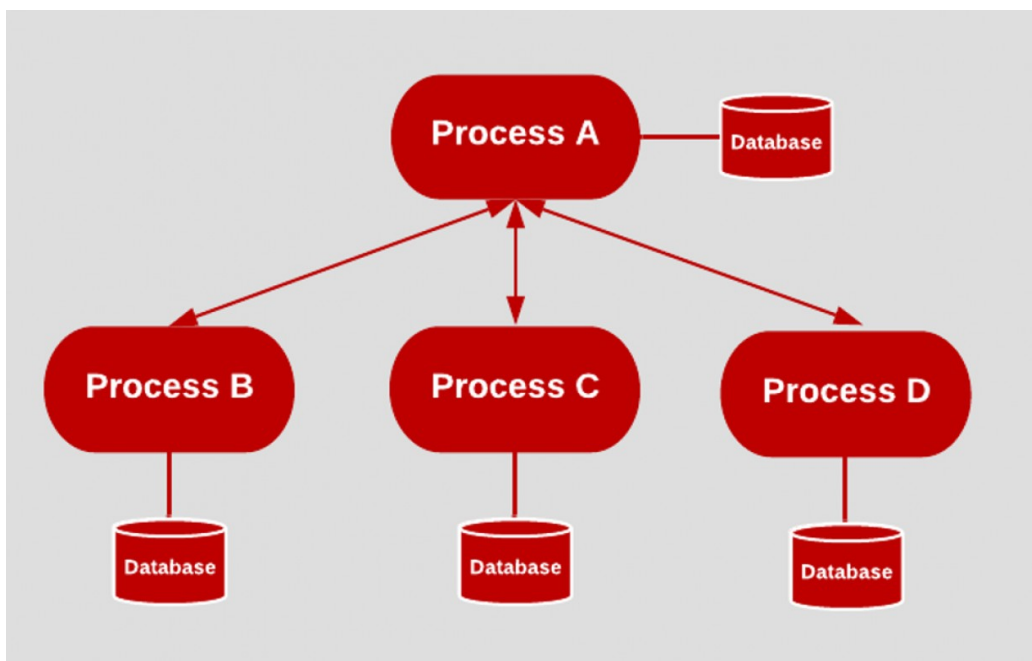
5. **Izraditi, pustiti i izvoditi (eng. *build, release and run*)** – strogo odvajanje faza izrade, puštanja i izvođenja

Peto načelo dijeli proces implementacije u tri strogo odvojene faze. Kodove se nakon jedne faze pregledavaju i testiraju pa tek onda prelaze u iduću fazu što bi značilo da su implementirani kodovi uvijek ispravni i dorasli svojoj namjeni. Primjerice, nije moguće napraviti izmjene u kodu za vrijeme izvođenja jer ne postoji način da se te izmjene vrate u fazu izrade.

6. **Procesi (eng. *processes*)** – izvođenje aplikacije kao jednog ili više procesa **bez stanja**

Šesto načelo procesa nalaže da se aplikacija izvodi u okruženju izvođenja (eng. *execution environment*) kao zbirka jednog ili više procesa bez stanja. To bi značilo da niti jedan proces neće pratiti stanje drugog procesa i da niti jedan proces ne prati informacije kao što su status sesije ili tijek rada. Konkretnije, nikada se ne pretpostavlja da će sve što je predmemorirano u memoriji ili disku biti dostupno budućem zahtjevu. Svi podaci koji se prikupljaju moraju

se pohraniti u neku prateću uslugu sa stanjem, a najčešće je to baza podataka.



Slika 9. Prikaz hijerarhije procesa u web aplikaciji
Izvor: redhat.com

7. Povezivanje portova (eng. *port binding*) – izvoz usluga povezivanjem portova

U tradicionalnom okruženju možemo pretpostaviti da različiti procesi upravljaju različitim funkcionalnostima koje su dostupne putem web protokola poput HTTP-a pa će se te aplikacije vjerojatno izvoditi na web poslužiteljima poput Apache web servera, no to se kosi s ovim načelom. Aplikacije izgrađene prema ovom načelu ne ovise o web poslužitelju te su potpuno samostalne i izvršavaju se samostalno. Osnovnoj aplikaciji može se dodati biblioteka web poslužitelja. Port na koji je aplikacija spojena mora biti pohranjen u konfiguraciji. To znači da aplikacija može čekati zahtjeve na definiranom portu, bilo da se radi o HTTP-u ili nekom drugom. Povezivanje portova jedan je od temeljnih uvjeta autonomnosti i samostalnosti mikroservisa.

8. Konkurencija (eng. *concurrency*) – povećanje kroz procesni model

Ovo načelo povezano je sa skaliranjem aplikacije i govori da je bolje implementirati više kopija umjesto povećavanja same aplikacije. Ukratko rečeno, načelo podržava horizontalno skaliranje aplikacije umjesto vertikalnog. Pridržavajući se ovog načela programeri mogu dizajnirati svoje aplikacije kako bi se nosili s različitim radnim opterećenjima dajući svakoj vrsti posla određeni tip procesa. Primjerice, HTTP zahtjevima može se baviti web proces, a dugotrajnim pozadinskim zadacima radnički proces. Podržavanje konkurencije znači da se

različiti dijelovi aplikacije mogu povećati kako bi zadovoljili trenutne potrebe. Model se pokazao veoma korisnim kad dođe vrijeme za skaliranje aplikacije tj. kad dođe do povećanja broja korisnika aplikacije.

9. Jednokratna upotrebljivost (eng. *disposability*) – maksimalno povećanje robusnosti brzim pokretanjem i elegantnim zaustavljanjem (eng. *graceful shutdown*)

Ovo načelo tvrdi da su procesi aplikacije za jednokratnu uporabu (eng. *disposable*), što znači da:

- procesi mogu započeti i završiti u trenutku
- procesi su robusni, odnosno dobro dizajnirani protiv iznenadnog kvara ili pada aplikacije
- procesi se elegantno prekidaju (eng. *graceful shutdown*), a to znači da softver, odnosno operacijski sustav, izvodi svoje zadatke i isključuje sustav na siguran način, za razliku od *hard shutdowna* gdje se računalo prisilno isključuje zbog prekida napajanja

Sve navedeno omogućuje brzo skaliranje, brzu implementaciju koda ili promjene konfiguracije, kao i robusnost proizvodne implementacije.

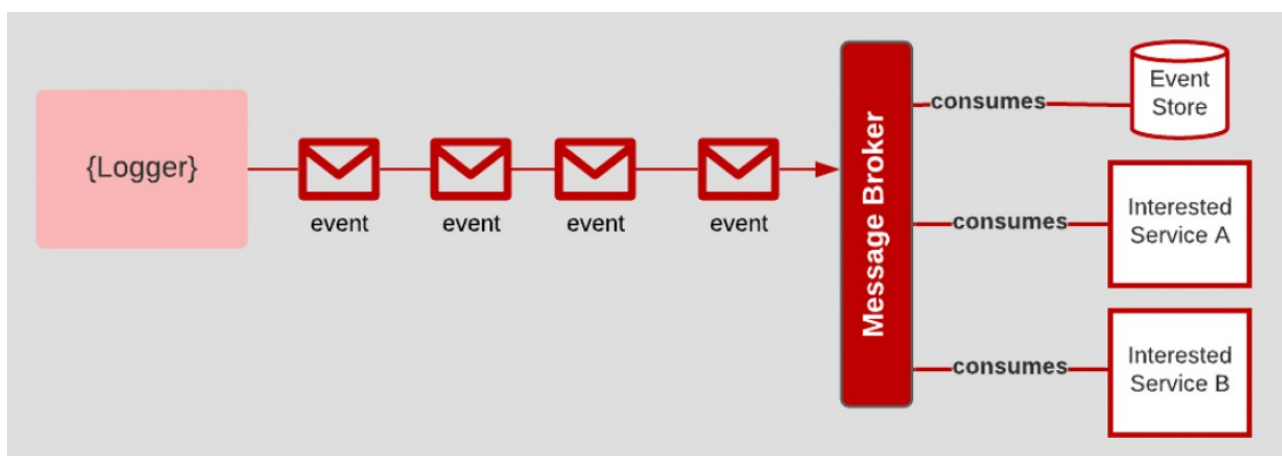
10. Jednakost razvoja i proizvodnje (eng. *Dev/prod parity*) – usklađivanje faza razvoja, postavljanja (eng. *staging*) i proizvodnje

Načelo jednakosti razvoja i proizvodnje predlaže da razlika između razvojnog i produkcijskog okruženja bude što manja. Tako se eliminira rizik od pojavljivanja grešaka u određenom okruženju, procesi čine organiziranim i jednostavnim te se podržava kontinuirano uvođenje (eng. *continuous deployment*). Kontinuirano uvođenje je proces u izdavanju softvera koji koristi automatizirano testiranje za provjeru jesu li promjene u bazi kodove ispravne i stabilne. Primjerice, idealno bi bilo koristiti istu bazu podataka u razvoju i proizvodnji,

11. Logovi (eng. *logs*) – gledanje na logove kao tijekove događanja

Prema načelu logova niti jedan okvir nije potpun bez efikasnog bilježenja logova koji mogu uvelike pomoći ispravljanju pogrešaka, ali i općoj provjeri ispravnosti koda. Načelo logova zagovara da se logovi tretiraju kao tijekovi događaja koji se usmjeravaju na zasebnu uslugu radi analize i arhiviranja. Logovi aplikacije tako će biti zapisani kao standardni izlazi, a okruženje za izvršavanje (eng. *execution environment*) će se pobrinuti za njihovo spremanje

i pohranu. Time se omogućuje veća fleksibilnost i sposobnost samoispitivanja ponašanja aplikacije. Logovi su ključna stavka za provjeru ponašanja aplikacije.



Slika 10. Prikaz logova kao tijekova događaja

Izvor: redhat.com

12. Admin procesi (eng. admin processes) – pokretanje administratorskih/upravljačkih zadataka kao jednokratnih procesa

Ovo načelo govori da bi aplikacije trebale pokretati administratorske ili upravljačke zadatke kao jednokratne procese u identičnom okruženju kao i regularni dugotrajni procesi aplikacije. Primjerice, to može biti jednokratni proces migracije baze podataka. Kod koji provodi ove operacije trebao biti isporučen s kodom aplikacije kako bi se izbjegli problemi sa sinkronizacijom.

Nakon ovog pregleda metodologiju 12 faktora može se ukratko definirati kao skup konkretnih i praktičnih tehnika za rješavanje problema koji se odnose na upravljanje konfiguracijom, implementacijom i dostupnošću aplikacije. Kada bi mene osobno netko pitao je li pri izradi web aplikacije potrebno slijediti sve ove principe da bi aplikacija bila pogodna za oblak vjerojatno bih rekao da to nije od presudne važnosti jer ovisi i o tome koja je uopće poslovna potreba same aplikacije. Međutim, trebalo bi se uvijek zapitati koji je razlog i posljedica odstupanja od pojedinog načela.

4. Primjer cloud web aplikacije u Flasku

4.1. Motivacija

S obzirom na to da o cloudu i cloud servisima nisam imao pozamašno znanje prije pisanja ovog završnog rada odlučio sam se baviti ovom tematikom upravo kako bih mogao nešto konkretno i korisno naučiti, a i cijela priča izgledala mi je veoma pitomo i simpatično. Ujedno mi je motivacija bila i stvaranje nečega konkretnoga, a u mojem slučaju funkcionalne i stvarno upotrebljive web aplikacije.

Već dulje vrijeme prije početka izrade aplikacije zanimalo me je na koji način se mogu izvlačiti podaci s raznih web lokacija. To je potaknulo moju želju za istraživanjem web spidera, odnosno web crawlera što se na kraju svelo na detaljnije proučavanje Beautiful Soup dokumentacije – Python biblioteke za grebanje, odnosno izvlačenje podataka iz HTML i XML datoteka. Zatim je uslijedila ideja o izradi aplikacije koja bi trebala izvlačiti podatke o cijenama proizvoda s nekoliko poznatijih hrvatskih web dućana koji prodaju informatičku opremu i koja na temelju tih cijena može dojaviti korisniku ako se dogodi pad cijene ispod željene cijene koju korisnik odredi za pretraživani proizvod. Daljnjim istraživanjem spoznao sam da bih tu ideju mogao realizirati koristeći *Amazon Web Services (AWS) Cloud service* i to konkretno koristeći AWS Lambda za pokretanje koda za obavijesti jednom dnevno i slanja email obavijesti korisniku što sam realizirao korištenjem *AWS Simple Notification Service (SNS)*. Naravno da bi cijela priča u pozadini bila jednostavna i razumljiva prosječnom korisniku web aplikacije trebalo je napraviti i grafičko sučelje, odnosno API (*Application Programming Interface*). To sam odlučio realizirati koristeći HTML i CSS.

Kako bih sve navedeno mogao objediniti u jednu funkcionalnu cjelinu istraživao sam i pronašao nešto za sebe – Flask. Flask je *microframework* napisan u Pythonu namijenjen izradi web aplikacija. Činjenica da je napisan u Pythonu posebno mi se svidjela s obzirom na to da je to programski jezik s kojim trenutno najbolje baratam. Svidjela mi se i mogućnost jednostavnog dodavanja Flask ekstenzija te poprilično jednostavno povezivanje i objedinjavanje *frontenda* i *backenda* što ga čini odličnim za razvoj web aplikacija.

4.2. Arhitektura i korištena tehnologija

Tijekom izrade spomenute aplikacije korišteno je nekoliko biblioteka, odnosno modula:

```
from flask import Flask, render_template, url_for, request
from bs4 import BeautifulSoup
import requests
from urllib.parse import urlparse
import concurrent.futures
import boto3
import uuid
```

Slika 11. Prikaz korištenih biblioteka i modula

Izvor: autor

1. Flask kao kostur i temelj aplikacije koji povezuje *backend* s *frontendom* preko Flask templatea
2. BS4 za izvlačenje podataka s web trgovina
3. Requests za pretraživanje web trgovina
4. Urllparse za izvlačenje dijelova web adrese nekih web trgovina kojih nije bilo moguće dohvatiti iz HTML koda
5. Concurrent.futures za ubrzavanje izvođenja scraping algoritma, odnosno threading
6. Boto3 za olakšani pristup AWS servisima izravno putem Python koda
7. uuid za automatsko kreiranje jedinstvenih ID-eva u DynamoDB bazi podataka i izradu naziva tema kod SNS servisa

4.2.1. Kreiranje rječnika

S obzirom na to da je zamišljeno da aplikacija pretražuje i izvlači cijene proizvoda na nekoliko web trgovina BeautifulSoupom, potrebno je bilo istražiti i pohraniti u rječnik nazive HTML tagova i klasa za svaku web trgovinu koji će se kasnije koristiti kod pretraživanja i izvlačenja podataka. Zbog estetike koda rječnik je spremljen kao zasebna skripta te kasnije importan u glavni kod.

```
webshop_dict = [
{'ime': 'H2 Shop', 'url': 'https://h2-shop.com/filterSearch?advs=true&cid=0&mid=0&vid=0&q=', 'naziv_pr': 'product-title',
'cijena_pr': 'price actual-price', 'link_tag': 'h2', 'link_class': 'product-title'},
{'ime': 'Instar', 'url': 'https://www.instar-informatika.hr/search.asp?upit=', 'naziv_pr': 'name',
'cijena_pr': 'numbers', 'link_tag': 'div', 'link_class': 'image'},
{'ime': 'Links', 'url': 'https://www.links.hr/hr/search?q=', 'naziv_pr': 'product-title',
'cijena_pr': 'price actual-price', 'link_tag': 'div', 'link_class': 'picture'},
{'ime': 'Mall.hr', 'url': 'https://www.mall.hr/trazenje?s=', 'naziv_pr': 'product-box-category_title',
'cijena_pr': 'product-price_price', 'link_tag': 'div', 'link_class': 'category-products_item'},
{'ime': 'PC Shop', 'url': 'https://www.pcshop.hr/search.asp?upit=', 'naziv_pr': 'prdocutname',
'cijena_pr': 'product-price', 'link_tag': 'div', 'link_class': 'product-image'},
{'ime': 'Tia mobiteli', 'url': 'https://www.tia-mobiteli.hr/katalog.aspx?izraz=', 'naziv_pr': 'product-title product-title-fixed-height',
'cijena_pr': 'product-price', 'link_tag': 'div', 'link_class': 'product-default inner-quickview inner-icon'}
]
```

Slika 12. Prikaz strukture korištenog rječnika

Izvor: autor

4.2.2. Algoritam za pretraživanje i izvlačenje podataka

Za idući korak važno je napomenuti da trenutno u rječniku postoje spremljeni jedinstveni dijelovi URL-ova za pretraživanje proizvoda za svaku web trgovinu. Sada se for petljom prolazi kroz svaki URL iz rječnika te se potom u varijablu *stranica* pohranjuju zahtjevi navedenim URL-ovima kojima se još pridodaje *item*. U ovoj fazi vrijednost varijable *item* unosila se preko Python konzole, dok će kasnije korisniku biti omogućen unos putem grafičkog sučelja web aplikacije, odnosno web preglednika. Sada se BeautifulSoupom grebu (eng. scrape) tj. pretražuju sve stranice te se u varijablu *soup* sprema njihov cjelokupni sadržaj, uključujući i sve HTML tagove. Za parsiranje se koristio lxml parser jer je, uspoređujući ga s ostalima, on najbrži parser kojeg BeautifulSoup podržava.

```
for shop in webshop_dict:
    stranica = requests.get(url=shop['url'] + item, headers = headers)
    soup = BeautifulSoup(stranica.content, 'lxml')
```

Slika 13. Prikaz prolaženja kroz sve URL-ove te izvlačenja sadržaja
Izvor: autor

S obzirom na to da se na krajnjem ispisu korisniku trebaju ispisati naziv trgovine, cijena proizvoda te link proizvoda bilo je potrebno za svaku web trgovinu, odnosno za svaki proizvod izvući navedeno. To je ostvareno kroz tri ugniježdene for petlje.

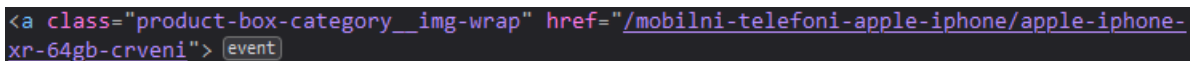
Prvo je započeto s izvlačenjem naziva proizvoda. Dakle, kad se pretražuju neki određeni proizvodi na nekoj web trgovini, ona vraća stranicu s rezultatima. Ovisno o izradi same web trgovine, to bude najčešće 12, 16, 20 ili 24 proizvoda poredanih po relevantnosti. Kako bi se posao olakšao te da se HTML tagovi koji prethode nazivu klase ne pretražuju ručno i svaki zasebno, svi mogući relevantni tagovi koji bi se mogli pojaviti pohranjeni su u polje *tagovi*. Zatim se for petljom prolazi kroz sve web trgovine i spremaju se nazivi proizvoda. Metoda *get_text()* koristi se kako bi se dobio čitljiv tekst i nazivi očistili od HTML tagova, dok se višak razmaka uklanja metodom *strip()* te se takve vrijednosti pohranjuju u varijablu *nazivSS*.

```
tagovi = ['a', 'p', 'span', 'h1', 'h2', 'h3', 'h3 title', 'div', 'ul', 'ol', 'ins', 'li', 'section', 'table', 'th', 'tb', 'label']

for naziv in soup.find_all (tagovi, class_ = shop['naziv_pr']):
    #print("nova proizv")
    nazivS = naziv.get_text()
    nazivSS = nazivS.strip()
    rezultati_proizvod.append(nazivSS)
    #print(nazivSS)
```

Slika 14. Prikaz pretraživanja i spremanja naziva proizvoda
Izvor: autor

Nakon naziva proizvoda bilo je potrebno izvući sve linkove. Za svaku web trgovinu u rječniku su izdvojeni tag i klasa za link na svaki proizvod te se for petljom prolazi svakim proizvodom. S obzirom na to da neke web trgovine nisu imale cjelovit link u HTML kodu, korišten je i *urllib.parse* modul kako bi se izvukla domena koja je često nedostajala. Primjer nedostatka domene (a i protokola) vidi se na slici 15, a riječ je bila o web trgovini mall.hr.

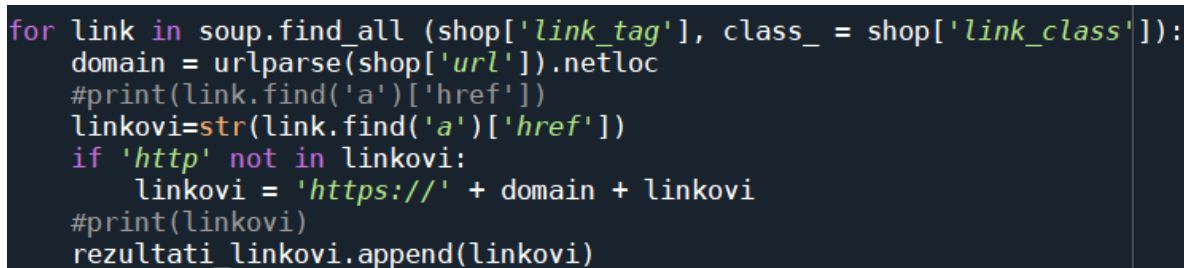


```
<a class="product-box-category__img-wrap" href="/mobilni-telefoni-apple-iphone/apple-iphone-xr-64gb-crveni">event
```

Slika 15: Prikaz nedostatka protokola i domene u HTML kodu web trgovine

Izvor: autor

Potom su u varijablu *linkovi* pospremljeni svi linkovi proizvoda, oni s domenom i oni bez. Zato je bila potrebna još jedna provjera koja će provjeravati sadrže li linkovi iz varijable *linkovi* domenu i protokol. Ako ne sadrže onda se oni dodaju i pospremaju nazad u varijablu *linkovi*.



```
for link in soup.find_all (shop['link_tag'], class_ = shop['link_class']):
    domain = urlparse(shop['url']).netloc
    #print(link.find('a')['href'])
    linkovi=str(link.find('a')['href'])
    if 'http' not in linkovi:
        linkovi = 'https://' + domain + linkovi
    #print(linkovi)
    rezultati_linkovi.append(linkovi)
```

Slika 16. Prikaz pretraživanja i spremanja linkova proizvoda

Izvor: autor

Postupak za pretraživanje i izvlačenje cijena proizvoda bio je nešto kompliciraniji od postupka za nazive i linkove. Za početak, cijena je potrebna kao *float* vrijednost, a ne *string*, kako bi se na kraju moglo uspoređivati odnose trenutnih cijena u trgovini i željenih cijena koje korisnik upisuje te temeljem toga korisniku slati obavijesti ako je trenutna cijena pala ispod željene. Da bi se dobiveni izraz pretvorio u *float* on mora biti valjano zapisan, a to u početku nije bio slučaj. Prvi korak bio je rješavanje znakova koji su bili višak, a to su bili znakovi za valutu kune. Pored toga, trebalo je ukloniti sve točke i zareze kako bi ostali samo brojevi. Potom je trebalo izdvojiti zadnja dva broja za lipe te ih pospremiti u varijablu *lipe*. S obzirom na to da je mall.hr bila jedina pretraživana web trgovina koja u svojim cijenama nije imala istaknute lipe i s obzirom na to da je prikaz cijene već bio ispravno formiran, za nju nije bio cilj oduzeti zadnje dvije znamenke jer bi se prikazivala netočna vrijednost te je stoga napravljena još jedna provjera koja za sve ostale trgovine formira cijenu u obliku: sve osim zadnje dvije znamenke + decimalna točka + *lipe*. Za kraj je dobivena cijena samo konvertirana u *float* vrijednosti te pospremljena u varijablu *cijenaSS*.


```

for cijena in soup.find_all (tagovi, class_ = shop['cijena_pr']):
    cijenaS = cijena.get_text()
    cijenaSS = cijenaS.strip()
    cijenaSS = str(cijenaSS).replace('\xa0Kn', ' ').replace(' kn', ' ').replace('Kn', '').replace('.', '').replace(',', '')
    lipe = cijenaSS[-2:]
    if shop['ime'] != 'Mall.hr':
        cijenaSS = cijenaSS[:-2]
        cijenaSS = cijenaSS + '.' + lipe
    cijenaSS = float(cijenaSS)
    rezultati_cijena.append(cijenaSS)

```

Slika 17. Prikaz pretraživanja i spremanja cijena proizvoda

Izvor: autor

Sada su tu pretraženi, izvučeni i pospremljeni nazivi proizvoda, linkovi i cijene u željenom obliku za svaki proizvod pretraživan prema korisnikovom unosu, ali se pojavio novi problem. Beautiful Soup poprilično je spor *scraper* kad su u pitanju veće količine podataka. Dobivanje rezultata sa svih 6 web trgovina može potrajati i po 30 sekundi, ovisno što se pretražuje i koliko svaka web trgovina ima relevantnih rezultata pretrage, što nije zadovoljavajući rezultat.

4.2.3. Uvođenje threadinga

Program funkcionira na sljedeći način. Nakon što korisnik upiše što želi pretražiti program odlazi na prvu web trgovinu, pretražuje taj proizvod te se dobivaju rezultati pretrage. Zatim se od tih rezultata prvo pretražuju i spremaju svi nazivi proizvoda. Nakon naziva pretražuju se i spremaju svi linkovi, a na posljetku se pretražuju i spremaju sve cijene proizvoda. Taj postupak ponavlja se za svaku web trgovinu. Iako se koristi lxml parser, to jednostavno nije dovoljno da algoritam funkcionira adekvatnom brzinom. Shodno tome uveden je *threading*.

Cjelokupan kod od prve for petlje prije definicije varijable *stranica* do kraja unutarnje for petlje za pretraživanje i izvlačenje cijena, pretvoren je u funkciju *scrape_ws* koja sadrži jedan argument *shop*. Zatim je uvezen modul *concurrent.futures* te se koristi njegov *ThreadPoolExecutor()* za uvođenje *threadinga*. Njegova klasa *executor* apstraktna je klasa koja pruža metode za asinkrono izvršavanje poziva. Ne smije se koristiti izravno, već kroz svoje potklase, a u konkretnom slučaju bila je potrebna potklasa *map*. Rezultat ove male, ali bitne promjene i nadopune koda ubrzavanje je trajanja izvođenja programa i do deset puta.

```

def scrape_ws(shop):

```

Slika 18. Prikaz zamijenjene glavne for petlje s definicijom funkcije s argumentom shop

Izvor: autor

```

with concurrent.futures.ThreadPoolExecutor() as executor:
    executor.map(scrape_ws, webshop_dict)

```

Slika 19. Prikaz implementacije threadinga

Izvor: autor

4.2.4. Ispis

Da bi se osiguralo ispravno funkcioniranje programa i kako ne bi nikad došlo do prekida programa zbog index errora korišteni su izrazi *try* i *except* jer se ponekad može dogoditi neočekivani problem zbog samih web trgovina. Kod jedne je otkrivena greška u procesuiranju upita te je prilikom pretrage određenih proizvoda, umjesto stranice koja vraća prikaz nekoliko relevantnih artikala, preusmjerilo na stranicu jednog konkretnog artikla i to bi prouzrokovalo index error jer nije bio jednak broj cijena, proizvoda i linkova.

```
try:
    for i, proizvod in enumerate(rezultati_proizvod):
        svi_proizvodi.append({
            'Naziv': proizvod,
            'Cijena(HRK)': rezultati_cijena[i],
            'Link': rezultati_linkovi[i]
        })
    print(svi_proizvodi)
    print(len(rezultati_cijena))
    print(len(rezultati_proizvod))
    print(len(rezultati_linkovi))
except IndexError:
    print('Ne možemo izvršiti pretragu za ovaj proizvod.')
```

Slika 20. Ispis naziva, cijene i linka za svaki pronađeni proizvod
Izvor: autor

4.2.5. Izrada Flask web aplikacije

Da bi se napisani programski kod mogao prilagoditi običnom korisniku korišten je Flask te njegovi predlošci (eng. *templates*). Izrada Flask aplikacije započela je s dodavanjem prvog dekoratora koji vodi na početnu stranicu web aplikacije. Zatim je definirana funkcija *index* koja vraća Flaskovu funkciju *render_template()* koja je korištena za prikazivanje sadržaja datoteke *index.html* korisniku na početnoj stranici web aplikacije.

```
@app.route('/')
def index():
    return render_template('index.html')
```

Slika 21. Dekorator za početnu stranicu web aplikacije
Izvor: autor

U aplikaciji su korišteni predlošci za renderiranje HTML koda. U Flasku je Jinja2 konfigurirana tako da se automatski izbjegavaju svi podaci koji su renderirani u HTML templateima. To znači da svi znakovi koje se unose, a koji bi se mogli pomiješati s HTML -om, poput „<” ili „>”, bit će zamijenjeni sigurnim vrijednostima koje izgledaju isto u pregledniku, a ne uzrokuju neželjeni efekt. Također postoje posebni razdjelnici (eng. *delimiter*) koji se koriste za razlikovanje Jinja2 sintakse

od statičkih podataka iz templatea. Primjerice, sve što se nalazi između znakova „{“ i „}” je izraz tj. varijabla koja se ispisuje u glavni dokument. Između oznaka „{%” i „%}” upisuju se naredbe poput *for* ili *if*, a za razliku od Python sintakse, blokovi naredbi označuju se na početku i na kraju, a ne samo uvlačenjem.

Za izradu dekoratora za pretragu korištena je HTTP metoda GET kao i za početnu (index) stranicu. Metodu GET nije potrebno posebno naznačivati jer je ona već unaprijed zadana metoda. Ujedno je metoda koja se najčešće koristi, a služi za slanje podataka na server u nešifriranom obliku. Da se radi o GET metodi, u praksi je vidljivo iz URL-a u kojem se raščlanjuje unos iz upita, odnosno to je onaj dio URL-a koji se u web pregledniku nalazi iza upitnika. U dekoratoru pretrage izvodi se ranije opisan scraping algoritam, a rezultati se, za razliku od ranije, više ne ispisuju preko Python konzole, već se vraćaju preko Flaskovih templatea u web pregledniku.

```
try:
    for i, proizvod in enumerate(rezultati_proizvod):
        svi_proizvodi.append({
            'Naziv': proizvod,
            'Cijena(HRK)': rezultati_cijena[i],
            'Link': rezultati_linkovi[i]
        })
    return render_template('pretraga.html', svi_proizvodi = svi_proizvodi)
except IndexError:
    return render_template('pretraga2.html')
```

Slika 22. Kod za ispis naziva, cijene i linka proizvoda na templateu
Izvor: autor

Na slici 23. prikazan je dio predloška *pretraga.html* koji prikazuje logiku ispisa informacija o proizvodima u web sučelju aplikacije. Prvo se prolazi svim proizvodima i ispisuju se svi nazivi i cijene. Ako se radi o linkovima, pored svakog proizvoda bit će dostupna veza na taj proizvod u web trgovini te će se vrijednosti varijable *link* klikom na gumb „Obavijest”, koji se nalazi pored ispisa informacija o tom konkretnom proizvodu proslijediti idućem dekoratoru: „/obavijesti” koji renderira template *obavijesti.html* na kojem korisnik upisuje tražene podatke.

```

<div class="proizvodi">
{% for dict_item in svi_proizvodi %}
<p>
    {% for key, value in dict_item.items() %}
        {% if key == 'Link' %}
            {{key}}:<a href="{{value}}">Klik</a>
            {% set link = value %}
            <form method="POST" action="/obavijesti">
            <input type="hidden" id="link" name="link" value="{{link}}">
            <input type="submit" value="Obavijest">
            </form>
        {% elif key == 'Cijena(HRK)' %}
            {{key}}:{{value}}
            {% set cijena = value %}
        {% else %}
            {{key}}:{{value}}
        {% endif %}
    {% endfor %}
{% endfor %}
</p>
</div>

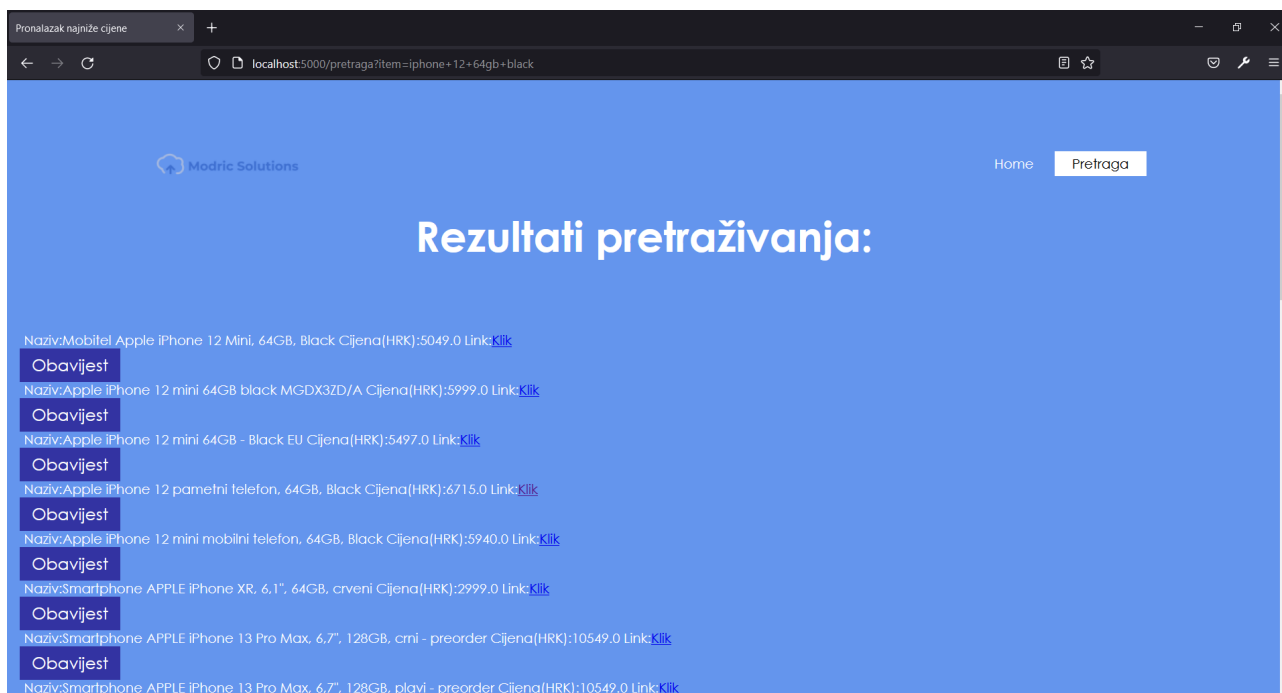
```

Slika 23. Logika ispisa s predloška „pretraga.html”

Izvor: autor

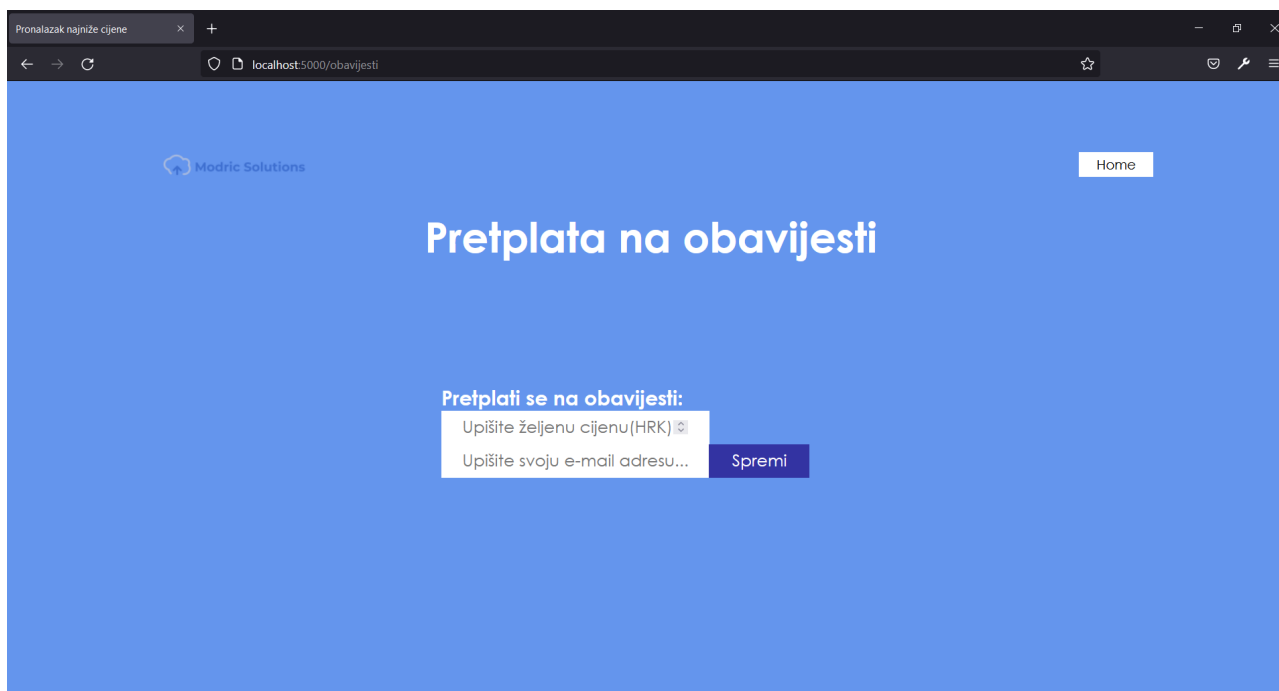
4.2.6. Dizajn sučelja web aplikacije

Naglasak pri izradi ove aplikacije stavljen je više na back end nego na front end. Front end je realiziran kao popratna pojava rada na back endu. Za izradu grafičkog sučelja web aplikacije korišteni su HTML i CSS jezici koji su korisniku prezentirani na web pregledniku posredstvom Flaskovih templatea. Kako u stvarnosti izgleda ispis rezultata pretraživanja i preplate na obavijesti u web pregledniku može se vidjeti na slikama 24 i 25.



Slika 24. Prikaz sučelja rezultata pretraživanja

Izvor: autor



Slika 25. Prikaz sučelja za pretplatu na obavijesti
Izvor: autor

Kao što je već napomenuto, HTML kodovi za svaku stranicu moraju biti pohranjeni u mapi „templates” jer ih Flaskova funkcija `render_template()` tamo traži. CSS kod i multimedijски sadržaji koji se koriste u aplikaciji pohranjuju se u mapi „static”.

4.2.7. Uvođenje Amazon web servisa

Amazonovi web servisi korišteni su kako bi aplikacija ostvarila svoj puni potencijal i bila u skladu s današnjim cloud trendovima.

- **EC2**

Amazon Elastic Compute Cloud, odnosno Amazon EC2 virtualni je poslužitelj te ujedno i najkorištenija usluga koju AWS nudi. EC2 omogućuje korisniku da učinkovito pokreće i upravlja instancama (virtualnim okruženjima) poslužitelja. Na EC2 virtualni poslužitelj postavljena je spomenuta i opisana Flask aplikacija.

- **DynamoDB**

DynamoDB je NoSQL baza podataka koja podržava ključ-vrijednost (eng. *key-value*) strukturu podataka. Skalabilna je, visoko dostupna, sigurna i brza baza podataka. Može obraditi više od 10 bilijuna zahtjeva dnevno. Pristup i konfiguracija izrazito su jednostavni. Korištena je za spremanje podataka potrebnih za pretplatu na obavijesti. U kodu je

implementirana koristeći boto3. Boto3 je komplet za razvoj softvera koji poboljšava i olakšava korištenje programskog jezika Python s AWS uslugama. Kada korisnik popuni obrazac za pretplatu na obavijesti, odnosno upiše željenu cijenu proizvoda i svoju e-mail adresu, u tablicu „obavijesti” spremaju se nasumično generiran ID, upisana e-mail adresa, proslijeđeni link proizvoda, upisana željena cijena, „emailPoslan” koji se standardno automatski postavlja na False vrijednost te TopicArn što predstavlja naziv teme za pretplatu na obavijesti.

```
if request.method == 'POST':
    email = request.form['email']
    zeljenaCijena = request.form['zeljenaCijena']
    link = request.form['link']

    response = clientSNS.create_topic(
        Name=str(uuid.uuid4()),
    )

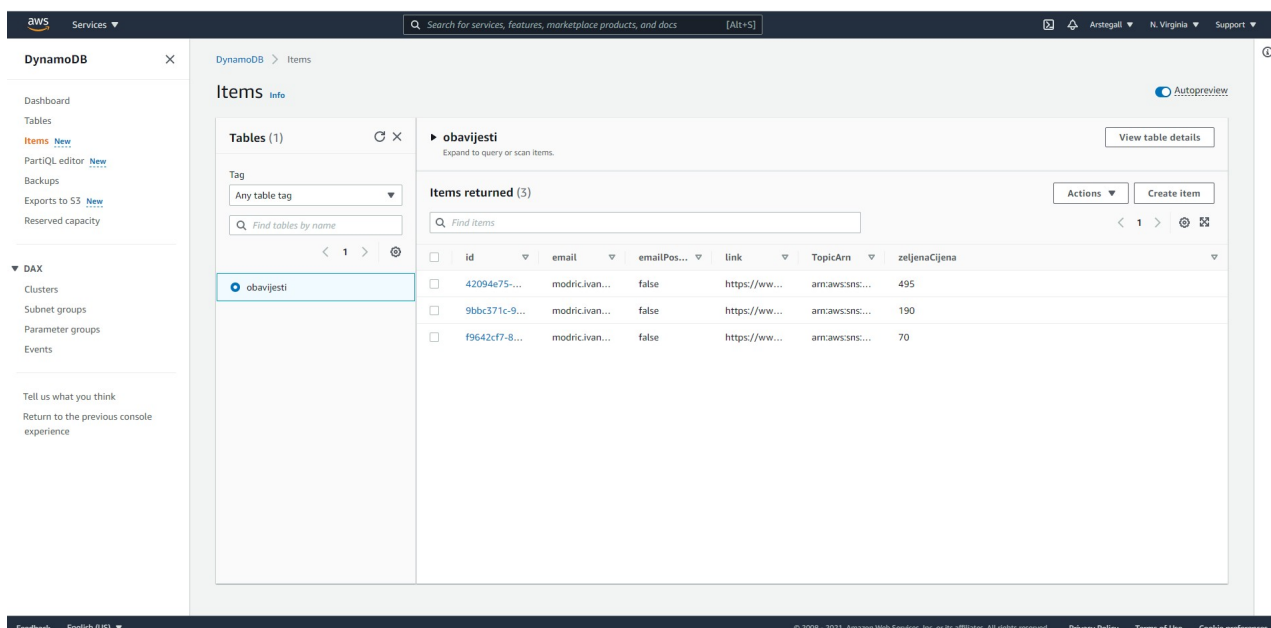
    clientDynamo.put_item(
        TableName='obavijesti',
        Item={
            'id': {
                'S': str(uuid.uuid4())
            },
            'email': {
                'S': email
            },
            'link': {
                'S': link
            },
            'zeljenaCijena': {
                'S': str(zeljenaCijena)
            },
            'emailPoslan': {
                'BOOL': False
            },
            'TopicArn': {
                'S': response['TopicArn']
            }
        })

    topic = resourceSNS.Topic(response['TopicArn'])

    topic.subscribe(
        Protocol='email',
        Endpoint=email,
    )
```

Slika 26. Kreiranje nove teme za obavijesti, zapisivanje podataka u tablicu „obavijesti” te slanje e-maila za pretplatu na obavijesti

Izvor: autor



Slika 27. Struktura tablice „obavijesti” na AWS konzoli
Izvor: autor

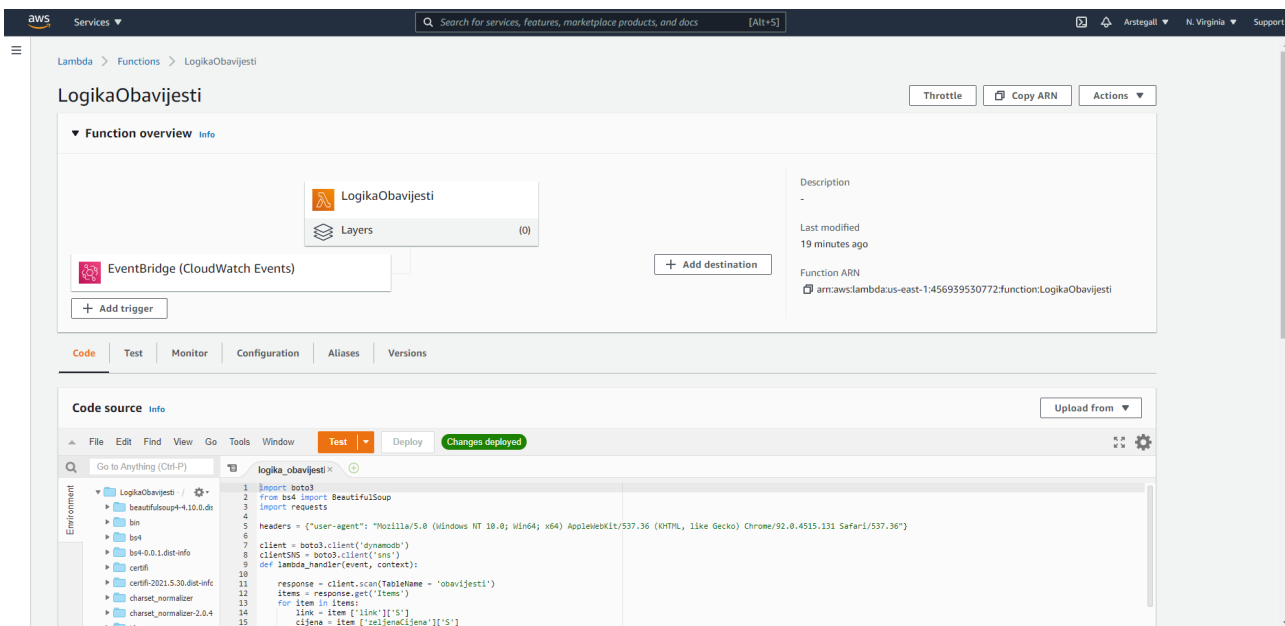
- **Simple Notification Service (SNS)**

SNS je usluga koja služi za razmjenu poruka između dvije aplikacije (A2A) ili između aplikacije i osobe (A2P). Postoji nekoliko protokola za slanje poruka poput Amazon SQS, Amazon Lambda, e-mail, SMS, HTTP i HTTPS protokola. Za potrebe aplikacije koristio se e-mail protokol.

Nakon što korisnik unese podatke za pretplatu na obavijesti kreira se nova tema za slanje obavijesti s nasumičnim nazivom, a nakon što se podaci upišu u DynamoDB, korisniku će se poslati e-mail za potvrdu pretplate na obavijesti, vidljivo na slici 26.

- **Lambda**

AWS Lambda računalna je usluga bez poslužitelja koja pokreće određeni kod kao odgovor na događaje i koja upravlja računalnim resursima. Lambda poziva programski kod samo kada je to potrebno te se sama automatski prilagođava kako bi podržala brzinu dolaznih zahtjeva. Na Lambdu se programski kod može veoma jednostavno prenijeti u ZIP formatu, a prilikom izrade opisane web aplikacije korištena je za logiku za slanje obavijesti te je u nju implementiran trigger - CloudWatch event koji je pokreće jednom dnevno.



Slika 29. Prikaz Lambde za logiku obavijesti na AWS konzoli
Izvor: autor

4.2.8. Izrada logike za slanje obavijesti

Korisniku se želi poslati obavijest kada cijena padne ispod željene cijene koju on upisuje u obrazac. To znači da je potrebno provjeriti cijenu konkretnog proizvoda. Naime, za izradu logike obavijesti procedura izvlačenje cijene jednog proizvoda nije se znatno razlikovala od prethodno opisane procedure za izvlačenje cijena s tražilice web trgovina. Razlikuju se samo nazivi HTML tagova i klasa pa je ponovno bilo potrebno istraživati HTML kod web trgovina.

Program za logiku obavijesti pokreće se jednom dnevno spomenutim CloudWatch eventom na Lambdi. Program prolazi kroz svaki zapis u bazi podataka i uspoređuje trenutnu cijenu proizvoda sa željenom cijenom tog proizvoda. Ako je trenutna cijena manja ili jednaka željenoj cijeni i ako je „emailPoslan” u bazi postavljen na False, korisniku se SNS-ovim e-mail protokolom šalje e-mail obavijest o padu cijene gdje se navodi link proizvoda i koja je trenutna cijena. Potom se u bazi podataka vrijednost „emailPoslan” postavlja na True kako se e-mail ne bi opet slao pri sljedećoj provjeri.


```

if cijena2 <= float(cijena) and emailPoslan == False:
    clientSNS.publish(
        TopicArn=topicArn,
        Message=f'Dobra vijest! Trenutna cijena Vašeg željenog proizvoda je pala! Sada iznosi {cijena2} kuna.\nPosjetite poveznicu: {link}',
        Subject='Pad cijene proizvoda!',
    )

    response = client.put_item(
        TableName='obavijesti',
        Item={
            'id': {
                'S': str(itemID)
            },
            'email': {
                'S': str(email)
            },
            'link': {
                'S': str(link)
            },
            'zeljenaCijena': {
                'S': str(cijena)
            },
            'emailPoslan': {
                'BOOL': True
            },
            'TopicArn': {
                'S': str(topicArn)
            }
        })

```

Slika 30. Dio logike za obavijesti i slanje e-maila
Izvor: autor

5. Zaključak

Završni rad započeo je prikazom evolucije dinamičkih web programskih jezika i okvira te njihove infrastrukture i arhitekture sve od CGI-a do Node.js-a i Flaska. Potom je predstavljena općenita arhitektura cloud web aplikacija podijeljena prema tri modela pružanja usluga u oblaku te su istaknute karakteristike i nezanemarive prednosti svakog od njih. Nakon toga objašnjena je metodologija 12 faktora te su navedene prednosti njezinog korištenja pri izradi web aplikacija u oblaku.

U radu su, također, opisane tehnologije i koncepti korišteni za izradu Flask web aplikacije te je detaljno opisana procedura samog pisanja programskog koda. Zaključak je da je Flask jedan zaista zgodan mikro okvir kojim se može napraviti jako puno na jednostavan način uz korištenje jednog od najmoćnijih programskih jezika današnjice – Pythona. Flask pruža jednostavnu implementaciju ranije isprogramiranog programskog koda. Flaskovi predlošci omogućuju da se sve ono što se zbiva u pozadini prikaže i grafički dočara korisniku u web pregledniku. Iako je sam po sebi generalno spor, web scraper BeautifulSoup ističe se lakoćom korištenja kod izvlačenja podataka s web stranica, a u mojem slučaju konkretno web trgovina. Amazonovi web servisi pružaju mnoge cloud usluge i funkcionalnosti koje su proces izrade dodatno pojednostavnile. Nisam se morao brinuti o potrebnoj infrastrukturi te su mi omogućile da aplikaciju napravim baš prema svojim željama i mogućnostima.

Ipak, esencijalni cilj ovog završnog rada bio je upravo izrada i prikaz jedne funkcionalne i u stvarnom životu korisne web aplikacije izrađene u Flask mikro okviru koja svoj puni potencijal postiže na Amazonovim web servisima. Glavni kod u Flasku postavljen je na AWS virtualni server EC2. Ako korisnik na početnu stranicu aplikacije u tražilicu upiše „Iphone 13 128GB Black” prikazat će mu se u kojim web trgovinama upisani proizvod može kupiti i po kojoj cijeni. Ako je korisnik zainteresiran za određenu trgovinu i proizvod, može se pretplatiti na obavijesti tako da klikne na gumb koji se nalazi pored svakog proizvoda. To će ga preusmjeriti na popunjavanje obrasca gdje će upisati željenu cijenu i svoju e-mail adresu nakon čega će se u DynamoDB pospremiti upisani podaci. Kod za provjeru cijena i slanje e-mail obavijesti nalazi se na AWS Lambdi te se provjera cijena vrši jednom dnevno korištenjem EventBridge triggera. Ako cijena proizvoda padne na korisnikovu željenu cijenu ili ispod nje te ako već ranije nije dobio e-mail koji dojavljuje pad cijene za taj proizvod poslat će mu se e-mail obavijest o padu cijene korištenjem AWS SNS servisa.

Postoji i plan za budućnost aplikacije, a to je daljnji rad i usavršavanje iste sa svim idejama koje mi budu nadolazile kroz vrijeme. Prvi budući korak, koji me već sad veseli, izrada je filtriranja proizvoda po cijenama i kategorijama. Aplikacija će u budućnosti biti doradivana i temeljem ostalih konstruktivnih ideja kojih se budem svojevremeno sjetio.

6. Literatura

- [1] <https://www.crummy.com/software/BeautifulSoup/bs4/doc/> zadnji pristup: 11.09.2021.
- [2] <https://docs.python-requests.org/en/latest/> zadnji pristup: 11.09.2021.
- [3] <https://boto3.amazonaws.com/v1/documentation/api/latest/index.html> zadnji pristup: 18.09.2021.
- [4] <https://flask.palletsprojects.com/en/2.0.x/> zadnji pristup: 18.09.2021.
- [5] <https://docs.aws.amazon.com/> zadnji pristup: 18.09.2021.
- [6] <https://aws.amazon.com/lambda/> zadnji pristup: 18.09.2021.
- [7] <https://docs.aws.amazon.com/ec2/> zadnji pristup: 18.09.2021.
- [8] <https://aws.amazon.com/sns/> zadnji pristup: 16.09.2021.
- [9] <https://aws.amazon.com/cloudwatch/> zadnji pristup: 16.09.2021.
- [10] <https://aws.amazon.com/dynamodb> zadnji pristup: 17.09.2021.
- [11] <https://12factor.net/> zadnji pristup: 15.09.2021.
- [12] <https://www.redhat.com/architect/12-factor-app> zadnji pristup: 15.09.2021.
- [13] <https://gorankrmpotic.eu/saas-vs-paas-vs-iaas/> zadnji pristup: 15.09.2021.
- [14] <https://medium.com/cuelogic-technologies/saas-paas-iaas-decoding-the-3-cloud-computing-service-models-25407ee1a568> zadnji pristup: 15.09.2021.
- [15] https://www.cs.ait.ac.th/~on/O/oreilly/perl/perlmut/ch10_01.htm zadnji pristup: 11.09.2021.
- [16] <https://www.geeksforgeeks.org/common-gateway-interface-cgi/> zadnji pristup: 11.09.2021.
- [17] <https://networkencyclopedia.com/common-gateway-interface-cgi/> zadnji pristup: 11.09.2021.
- [18] <https://rubyonrails.org/> zadnji pristup: 13.09.2021.
- [19] <https://pcchip.hr/softver/korisni/ruby-on-rails/> zadnji pristup: 13.09.2021.
- [20] <https://www.php.net/> zadnji pristup: 12.09.2021.
- [21] <https://metacpan.org/pod/CGI> zadnji pristup: 13.09.2021.
- [22] <https://jinja.palletsprojects.com/en/3.0.x/> zadnji pristup: 17.09.2021.

- [23] <https://www.fullstackpython.com/flask.html> zadnji pristup: 16.09.2021.
- [24] <https://promyze.com/understanding-12-factors-app/> zadnji pristup: 15.09.2021.
- [25] <https://docs.python.org/3/> zadnji pristup: 17.09.2021.
- [26] <https://docs.djangoproject.com/en/3.2/> zadnji pristup: 16.09.2021.
- [27] <https://djangostars.com/blog/why-we-use-django-framework/> zadnji pristup: 16.09.2021.