Univerzitet Singidunum

Departman za poslediplomske studije

Master Rad

Servisno-orijentisani razvoj aplikacije za predikciju cene zlata

|  |  |
| --- | --- |
| Mentor | Student: Mihailo Šević |
| Prof. Dr Angelina Njeguš | Broj indeksa 2018440018 |

Beograd 2021



**Univerzitet Singidunum**

**Departman za poslediplomske studije**

Master akademski program

Softversko i informaciono inženjerstvo

**Master rad**

Servisno-orijentisani razvoj aplikacije za predikciju cene zlata

|  |  |
| --- | --- |
| Mentor: | Student |
| Prof. Dr Angelina Njeguš | Mihailo Šević , 2018440018 |

Beograd 2021

**Apstrakt**

*Poslednjih godina razvojem tehnologije se sve više uvidja potreba za mašinskim učenjem i primenom mašinskog učenja u različitim oblastima poslovanja. Mašinsko učenje postaje sve važnije za različite predikcije dešavanja, posebno na finansijskom tržištu koje je veoma osetljivo na promene, kako prirodne, tako i društvene. Kriza Covid-19 virusa je dovela do toga da ljudi „beže“ u štednju u nečemu opipljivijem od svetskih valuta, a to je zlato. Ta vrsta premeštanja finansijkih sredstava je dovela do naglog poskupljenja cene zlata i sve predikcije radjene na osnovu modela linearne regresije su pale u vodu. Stoga, da bi se pomoglo u donošenju finansijskih odluka u smislu ulaganja u različite valute, osmišljena je aplikacija koja će da nam daje predikciju sutrašnje cene zlata. Značaj ove aplikacije je lakoća korišćenja za ljude iz netehničkih nauka koji nisu imali priliku da se sreću sa kreiranjem i odabirom modela za predikcije, koji sada jednostavno mogu doći do informacije o predvidjenoj ceni samo uz par klikova. Takodje, sekundarna funkcionalnost ove aplikacije je u tome što se ne mora raditi samo predikcija cene zlata, nego i predikcije cena bilo kojeg finansijskog instrumenta, derivata, robe... Praktično se može izvršiti bilo koja predikcija ukoliko postoje odgovarajući ulazni podaci.*

*Za kreiranje ovakve aplikacije korišćena je servisno orijentisana arhitektura unutar koje se nalaze četiri servisa: broker koji predstavlja sponu izmedju ostalih servisa, frontend servis koji prikazuje podatke korisnicima aplikacije, prediktor servis koji priprema, čuva i dostavlja podatke vezane za samu predikciju i modelator servis koji se bavi samim mašinskim učenjem i predikcijama.*

**Abstract**

*In recent years, the development of technology has increasingly recognized the need for machine learning and the application of machine learning in various areas of business. Machine learning is becoming increasingly important for various predictions of events, especially in the financial market, which is very sensitive to changes, both natural and social. The crisis of the Covid-19 virus has led to people to save in something more tangible than world currencies, and that is gold. This type of transfer of funds led to a sharp rise in the price of gold and all predictions made on the basis of the linear regression model failled. Therefore, in order to help make financial decisions in terms of investing in different currencies, an application has been designed that will give us a prediction of tomorrow's gold price. The significance of this application is the ease of use for people from non-technical backgrounds who have not had the opportunity to encounter the creation and selection of models for predictions, who can now easily get information about the estimated price with just a few clicks. Also, the secondary functionality of this application is various price predictions can be done, not only the gold price prediction, but also the prediction of price of any financial instrument, derivatives, commodities ... Practically any prediction can be made if there is appropriate data input.*

*To create such an application, a service-oriented architecture was used within which there are four services: a broker that represents a link between other services, a frontend service that displays data to application users, a predictor service that prepares, stores and delivers data related to prediction and a modeler service that deals with machine learning and predictions themselves.*

**Key words:** SOA, machine learning, price prediction

Sadržaj

[1. Uvod 1](#_Toc69342145)

[2. Servisno orijentisana arhitektura 2](#_Toc69342146)

[2.1. Servisi 5](#_Toc69342147)

[2.2. Komunikacija, protokoli i hosting 5](#_Toc69342148)

[3. Aplikacija 7](#_Toc69342149)

[3.1. Predikcija cene zlata 7](#_Toc69342150)

[3.2. Prethodna istraživanja predkcije zlata 8](#_Toc69342151)

[3.3. Korišćene tehnologije 10](#_Toc69342152)

[3.3.1. Docker 11](#_Toc69342153)

[3.3.2. React 11](#_Toc69342154)

[3.3.3. Express 12](#_Toc69342155)

[3.3.4. Socket.io 12](#_Toc69342156)

[3.3.5. Got 12](#_Toc69342157)

[3.3.6. Flask 13](#_Toc69342158)

[3.3.7. Apscheduler 13](#_Toc69342159)

[3.3.8. Pandas 13](#_Toc69342160)

[3.3.9. Scikit-learn 14](#_Toc69342161)

[3.4. Korišćeni servisi 14](#_Toc69342162)

[3.4.1. Broker 14](#_Toc69342163)

[3.4.2. Frontend 15](#_Toc69342164)

[3.4.3. Prediktor 16](#_Toc69342165)

[3.4.4. Modelator 16](#_Toc69342166)

[3.5. Arhitektura 17](#_Toc69342167)

[4. Zaključak 19](#_Toc69342168)

[4.1. Budući razvoj 19](#_Toc69342169)

[5. Literatura 21](#_Toc69342170)

[6. Prilozi 25](#_Toc69342171)

[6.1. Github 25](#_Toc69342172)

[6.2. Struktura fajlova 25](#_Toc69342173)

[6.3. Van servisni fajlovi 26](#_Toc69342174)

[6.3.1. README.md 26](#_Toc69342175)

[6.3.2. LICENSE 26](#_Toc69342176)

[6.3.3. docker-compose.yml 26](#_Toc69342177)

[6.3.4 .gitignore 27](#_Toc69342178)

[6.4. Fajlovi Broker Servisa 27](#_Toc69342179)

[6.4.1. broker/sevices/forntend/service.js 27](#_Toc69342180)

[6.4.2. broker/sevices/modelator\_d3\_rw/service.js 27](#_Toc69342181)

[6.4.3 broker/sevices/predictor/service.js 28](#_Toc69342182)

[6.4.4 broker/index.js 28](#_Toc69342183)

[6.4.5. broker/Dockerfile 29](#_Toc69342184)

[6.4.6. broker/package.json 29](#_Toc69342185)

[6.5. Fajlovi Frontend servisa 30](#_Toc69342186)

[6.5.1 frontend/package.json 30](#_Toc69342187)

[6.5.2. frontend/Dockerfile 30](#_Toc69342188)

[6.5.3. frontend/src/App.js 30](#_Toc69342189)

[6.6 Fajlovi Modelator servisa 32](#_Toc69342190)

[6.6.1. modelator\_d3\_rw/app.py 32](#_Toc69342191)

[6.6.2. modelator\_d3\_rw/modelator.py 32](#_Toc69342192)

[6.6.3. modelator\_d3\_rw/models.py 33](#_Toc69342193)

[6.6.4. modelator\_d3\_rw/responder.py 34](#_Toc69342194)

[6.6.5. modelator\_d3\_rw/rolling\_window.py 34](#_Toc69342195)

[6.6.6. modelator\_d3\_rw/splitter.py 34](#_Toc69342196)

[6.6.7. modelator\_d3\_rw/standard\_model.py 35](#_Toc69342197)

[6.6.8. requirements.txt 35](#_Toc69342198)

[6.6.9. Dockerfile 35](#_Toc69342199)

[6.7.Falovi Predictor servisa 36](#_Toc69342200)

[6.7.1. predictor/app.py 36](#_Toc69342201)

[6.7.2. predictor/predictor.py 36](#_Toc69342202)

[6.7.3. predictor/vars.py 38](#_Toc69342203)

[6.7.4. predictor/Dockerfile 38](#_Toc69342204)

[6.7.5. predictor/requirements.txt 38](#_Toc69342205)

# 1. Uvod

Ovaj rad govori o primeni servisno orijentisane arhitekture na mašinsko učenje i finansijske predikcije.

Svaka aplikacija, softver, ima unutrašnju organizaciju. Tu organizaciju nazivamo softverskom arhitekturom i najčešće se dešava da krajnji korisnici aplikacije nisu ni svesni postojanja te arhitekture ili bilo kojeg procesa koji se dešava u pozadini same aplikacije.

U ovom radu je za izradu aplikacije za predikciju kretanja cena zlata korišćena servisno orijentisana arhitektura, što znači da su komponente aplikacije organizovane u zasebne servise koji bi mogli da se tretiraju kao zasebne aplikacije. Za realizaciju aplikacije su korišćena četiri servisa/celine:

- Broker (servis koji predstavlja sponu izmedju svih ostalih servisa)

- Prediktor (kontrola toka aplikacije, proračuna)

- Modelator (servis za mašinsko učenje)

- Frontend (prezentacioni sloj)

Same celine su vrlo nezavisne jedna od druge, a medjusobno imaju interakciju preko API-ja kroz htttp protokol. Ovakva nezavisnost medjusobna komunikacija omogućavaju da se servisi mogu nezavisno koristiti, bez ili sa jako malim unutrašnjim izmenama. Takodje, ovakvom arhitekturom se omogućava da se servisi nalaze na različitim fizičkim lokacijama i da imaju različite oblike hostinga (serverless).

Aplikacija predstavlja alat za predikciju sutrašnje cene zlata. Kreiranje ove aplikacije je odabrano jer predstavlja koristan alat za pomoć u donošenju odluka vezanih za ponašanje na finansijskom tržištu. Takodje, odabrana je jer je ovakve odluke veoma teško donositi i u normalnim uslovima, a još teže u periodima velikih nestabilnosti poput trenutne situacije sa COVID-19 virusom. Primena mašinskog učenja za predikciju cena zlata gde se uzimaju u obzir parametri COVID-19 virusa je već proverena sa izuzetno dobrim rezultatima sa preko 95% tačnosti u predikciji (Stokanović Šević 2020.).

Ovakve metode predikcije, iako su vrlo precizne, nisu baš jednostavne za upotrebu jer je potrebno znanje programiranja da bi se došlo do rezultata. Aplikacija koja je izrađena u ovom radu otvara mogućnosti onima koji znaju kakva vrsta predikcija im treba u smislu odabira parametara i finansijskih derivata za predikcije, ali nisu vični programiranju. Ona olakšava pristup različitim modelima mašinskog učenja, sa tim da daje i prikaz rada modela na primeru predikcije sutrašnje cene zlata.

# 2. Servisno orijentisana arhitektura

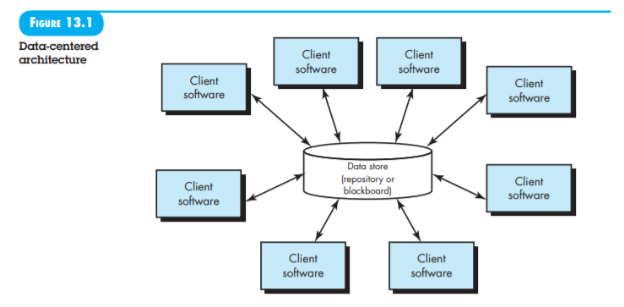
Softverska arhitektura je fundamentalna struktura koja diktira razvoj softvera. Svaka softverska arhitektura se sastoji od elemenata i njihovih medjusobnih odnosa.(Clements et.al., 2010)

Značaj arhitekture je u tome da ona predstavlja nacrt softvera koji želimo da napravimo. Kao i nacrt za izgradnju kuće, na osnovu arhitekture možemo odrediti karakteristike i pre nego što sama izgradnja počne. Prilikom dizajniranja arhitekture možemo videti poteškoće koje bi mogle da nas čekaju u budućnosti i da promenimo dizajn, te da predupredimo problematične situacije ili bar da budemo spremni kad dodje do njih.

Softverska arhitektura ko pojam se prvi put pominje šezdesetih godina prošlog veka, a sporadično do devedesetih. U tom periodu akcenat u rešavanju kompleksnih problema je bio na strukturama podataka i različitim algoritmima. Začeci softverske arhitekture se mogu naći u istraživanjima Edsger Dajkstra 1968 i Davida Parnasa 1970 kada je ustanovljeno da je struktura softvera bitna i da je izuzetno bitno da je ta struktura dobro definisana. (Garlan & Shaw, 1993)

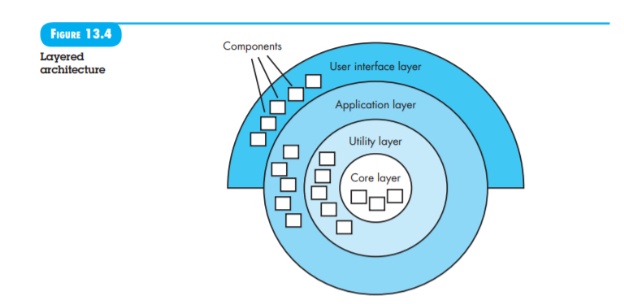
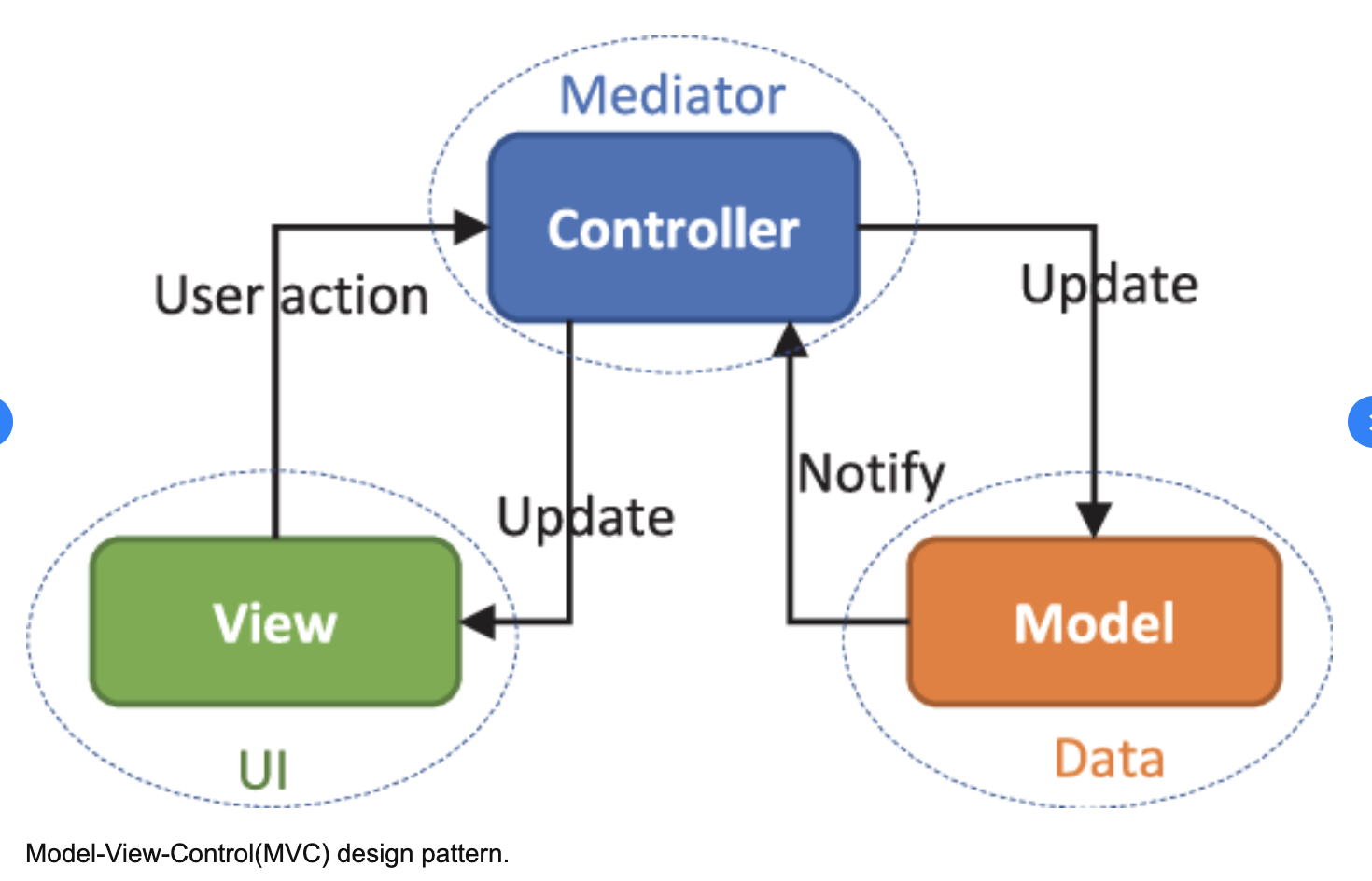
Devedesetih godina prošlog veka dolazi do buma u razvoju softvera sa dot com balonom. Sa ubrzanim razvojom softvera dolazi do diferenciranja nekoliko karakterističnih arhitektura.

Jedna od prvih arhtektura je arhitektura orijentisana ka podacima. To znači da su podaci bili skladišteni na jednom mestu, a različiti klijenti su mogli da vrše operacije nad tim podacima. Ova arhitektura pripada periodu mainfrejmova i skladišta podataka. (Pressman & Maxim, 2014)



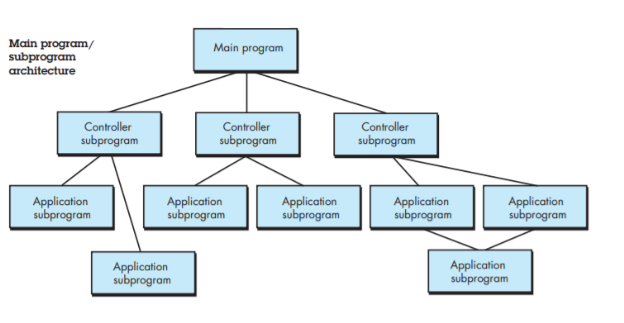
Slika 1. Prikaz arhitekture orijentisane podacima

Najpopularniji arhitektonski pristup, koji je i dalje izuzetno popularan u malim do srednjim softerima je MVC(model, view, controller) šablon. Ova troslojna arhitektura je postala popularna iz nekoliko razloga medju kojima su objektno orijentisano programiranje i brza postavka inicijalnog rešenja. Brza postavka je omogućena raznim frejmvorcima koji imaju već predefinisane sržne komponente. Objektno orijentisano programiranje dozvoljava nasledjivanje sržnih elemenata što dalje smanjuje vreme izrade i povećava kvalitet samog softvera. (Pressman & Maxim, 2014)



Slika 2. Prikaz MVC arhitekture

Nešto pre MVC arhitekture korišćena je pozivna arhitektura gde glavni program poziva druge podprograme radi izvršavanja zadataka. Ova arhitektura se može u neku ruku smatrati pretečom servisno orijentisane arhitekture jer je svaki podprogram specijalizovan za odredjenu radnju, slično kao što svaki servis obavlja zadatke odredjene poslovne celine. (Pressman & Maxim, 2014)



Slika 3. Prikaz podprogramske arhitekture

Servisno orijentisana arhitektura je vrlo mlad vid arhitekture koji se pojavljuje krajem prve decenije dvadeset i prvog veka. Arhitektura je nastala razvojem web servisa, koji su postali upotrebljivi oko 2010e godine sa unapredjenjem internet brzina.

Kao što je već napomenuto, svaki softver ima dizajn koji može koristiti jednu ili više arhitektura, a u konkretnom slučaju odabrana je servisno orijentisana arhitektura zbog svoje fleksibilnosti.

Servisno orijentisana arhitektura je nezavisna od tehnologija, proizvoda i dobavljača(vendor).

Ova aritektura predstavlja takav međusobni odnos logičkih/funkcionalnih celina da odlično prati poslovnu logiku očekivanog rezultata aplikacije. (Bell, 2008)

Svaka celina se naziva servisom i obavlja svoju ulogu ili nezavisno ili delegirajući odredjene zadatke drugim servisima koji su sposobni za izvršenje takvih zadataka. Prednosti ovakve arhitekture su konciznost koda i mogućnost testiranja pojedinačnih elemenata(servisa), laka skalabilnost, diverzitet izbora tehnologija servisa kao i fleksibilna distributivnost. (Dragoni et.al., 2017)

Mana ovakve arhitekture je u tome što se lako dolazi u situaciju preteranog broja servisa koje je teško upratiti što vodi do servisa koji rade iste stvari i medjusobnog preplitanjima u izvršavanju. Takođe, mana može biti sporost sistema u slučaju da je komunikacija između sistema otežana.

Prilikom dizajniranja servisno orijentisane aplikacije/sistema poželjno je voditi se SOA manifestom (*SOA Manifesto*, 2009) koji se sastoji od šest postulata:

* Poslovna vrednost je bitnija od tehničke strategije
* Strateški ciljevi su bitniji od projektno specifičnih benefita
* Sopstvena interoperabilnost nad specifičnom integracijom
* Deljeni servisi su bitniji od specifičnih servisa
* Fleksibilnost je bitnija od optimizacije
* Evolutivno usavršavanje je bitnije nego insistiranje inicijalnoj perfekciji.

Svaki servis predstavlja jednu zaokruženu poslovnu celinu i moše igrati jednu od tri uloge:

* Servis provajder, predstavlja sam servis i dužnost mu je da definiše koji će interfejs biti dostupan brokeru i pod kakvim uslovima(odobrenjima).
* Broker ili registar servisa predstavlja komponentu preko koje se vrši interakcija sa servisima. Javni broker je dostupan svuda i svima , dok interni broker vrši komunikaciju izmedju servisa. Ova dva brokera mogu biti spojeni u jedan broker opšteg tipa.
* Potraživač ili korisnik servisa, predstavlja komponentu koja od brokera zahteva pristup odredjenom servisu.

(Rotem-Gal-Oz, 2012)

## 2.1. Servisi

Servisi su mikroaplikacije koje obavljaju specifične zadatke. Njihova interna organizacija može biti raznovrsna, od jednostavnih funkcija do kompleksnih podprograma koji se sastoje od više slojeva. Takodje, servisi se ponašaju kao crne kutije sa predefinisanim interfejsima što nam pomaže da ne moramo previše razmišljati o samoj strukturi servisa.

Prvi servisi su nastali nezavisno od arhitekture, to su takozvani web servisi. U pitanju su web aplikacije koje umesto korisničkog interfejsa vraćaju podatke u vidu XML ili JSON stringova.

Sama tehnologija u kojoj je servis realizovan je potpuno nebitna dokle god ima jasno definisan interfejs i brzo i kvalitetno izvršava svoje zadatke.

Servisi imaju mnogih prednosti zbog svoje atomičnosti. To znači da jedan servis u sebi sadrži sve što je neophodno da bi se zadaci izvršavali. Jedan servis je moguće koristiti od strane više različitih aplikacija istovremeno čime se smanjuje vreme razvoja tih aplikacija, ali se smanjuje i vreme potrebno za održavanje jer je potrebno održavati samo jedan servis.

Pošto su servisi pravljeni da prate poslovnu logiku i sami servisi mogu predstavljati uslugu koju firma nudi svojim klijentima što nam omogućava da napravimo aplikaciju u kojoj se dodatne mogućnosti dodatno naplaćuju. (Bell, 2008)

## 2.2. Komunikacija, protokoli i hosting

Komunikacija izmedju servisa se izvršava preko preko mreže koristeći http protokol. Http je skraćenica za Hypertext Transfer Protocol, dakle to je protokol za razmenu jasno strukturisanih poruka. (Leach et.al., 1999)

Najčešća komunikaciona arhitektura je REST ili Representational state transfer. REST arhitektura koristi podset http protokola i detaljno je definisana. Pod detaljnijim definisanjem se misli na dodatno definisanje strukture samog tela poruka, puna upotreba http metoda (GET, POST, DELETE...) kao i odgovora u skladu sa promenom stanja što je najčešće prikazano http statusnom porukom. (Lange, 2016)

REST arhitektura je dobila na popularnosti zbog svoje jednostavnosti i brzine. Prethodna, SOAP arhitektura je deskriptivnija i koristi XML format za razmenu poruka, što je u suštini i ključna razlika izmedju SOAP i REST arhitektura. SOAP pristup sa slanjem poruka u XML formatu je previše deskriptivan i time zauzima previše resursa, pogotovo kad se komunikacija vrši preko ograničenih mreža poput mobilne. Iz tog razloga je REST uzla maha u poslednje dve decenije. Iako REST predstavlja veliki napredak u odnosu na SOAP i dalje postoje problemi vezani za REST. Najveća dva problema su Over i Under fetching. Overfetching znači da se od servera na zahtev dobija više informacije nego što je potrebno, a Underfetching znači da je potrebno napraviti više zahteva da bi se prikazala celokupna slika.

Kao rešenje za prethodna dva probelma je 2015 napravljen GraphQL koji kroz jedan zahtev dobija sve potrebne informacije, a opet ne koristi komplikovane strukture poprut XML-a. Kao i svaki protokol i GraphQL ima svojih mana, prevashodno što ne postoji način da se naglasi verzija. (DA-14, 2018)

Komunikacija ne mora da bude preko http protokola pogotovo kad je potrebna uzajamna asinhrona komunikacija izmedju servera i klijenta. Ako je potrebna ovakva komunikacija može se koristiti web-socket protokol koji predstavlja full duplex komunikacioni kanal izmedju dve strane. Web soket protokol je standardizovan 2011e i podržan je od strane većine novijih web pregledača. Prednost ovog protokola je u tome što se klijetu mogu slati informacije iako ih nije tražio. Ova mogućnost je od ključnog značaja za čet aplikacije/servise i vrlo korisna za real-time obaveštenja. Web soket protokol se moše koristiti za komunikaciju izmedju klijenta i servisa kao i izmedju samih servisa, isto kao i http. (Peter Lubbers & Frank Greco, 2020)

Pošto je komunikacija izmedju servisa standardizovana i koristi standardne protokole, informacije se prenose preko mreže između nezavisnih mašina. Ova karakteristika omogućava veliku fleksibilnost u hostingu tako da statički elementi sistema poput frontenda mogu biti hostovani na CDN (content delivery network) koji čuvaju kopije blizu klijenata tako da se smanjuje inicijalno vreme učitavanja aplikacije. Logički elementi se mogu izvršavati na zasebnim serverima ili kao servisi na serverless sistemima. Odabir same organizacije hostinga i organizacije lokacije servisa su kompleksna pitanja koja dosta zavise od same aplikacije i potražnje odredjenih servisa aplikacije. (Laws & Goodfellow, 2018)

Servisno orijentisana arhitektura se uglavnom koristi za kompleksne sisteme(aplikacije), ali se može koristiti delom u manjim projektima gde se odredjeni servisi samo inkorporiraju spram potreba.

# 3. Aplikacija

Mašinsko učenje i predikcije su vrlo popularne tematike koje nalazee sve više oblasti primene. Iako imaju veliku primenjivost, mali je broj ljudi u interesnim oblastima kojima je jasno i koji imaju umeća kako da primene metode mašinskog učenja. Ovaj problem je naročito izražen u ekonomskim disciplinama. Uglavnom ljudi koji se bave ekonomijom slabo razumeju programiranje, a mašinsko učenje još manje. Ovaj razlog je primarni pokretač za pravljenje aplikacije ovog master rada.

Aplikacija je napravljena tako da omogući lakši pristup mašinskom učenju osobama koje nemaju umeća da sami naprave predikcije. Takodje, pokazuje primenjivost na konkretnom problemu predikcije cene zlata što je značajan podatak za osobe koje se bave investiranjem poput brokera i investicionih savetnika.

Najbitniji aspekt prilikom izrade aplikacije je bila višestruka primena servisa. Svaki servis je napravljen ili da ima univerzalnu funkciju poput brokera i modelatora ili da se lako može proširiti novim mogućnostima poput prediktora. Na ovaj način napravljena je aplikacija da uz odgovarajuć zahtev svako može da dodje do rezultata predikcije bez obzira na poreklo i sadržaj podataka i da, ako se javi potreba, se proširi predikcija na druge finansijske derivate,a ne samo zlato.

Iz želje da se napravi aplikacija koja se može lako menjati spram potreba klijenata i koja može da pruži uslugu koja će biti primenjiva na raznovrsnim podacima, odlučeno je da se koristi servisno orijentisana arhitektura jer se postulati poklapaju sa planom aplikacije.

## 3.1. Predikcija cene zlata

Zlato kao plemeniti metal je oduvek imalo vrednost koju je bilo moguće zameniti za robu ili uslugu. Preferirano i likvidno je zato sto je relativno lako prenosivo, ne gubi svoje karkteristike vremenom (sjaj, masu, oblik), nije široko dostupno jer je potrebno dosta truda, a i sreće da bi se u prirodi našle i najmanje količine.

Zahvaljujući svojim karakteristikama našlo je primenu u prvom novcu u vidu kovanica. Kasnije je uveden novac u papiru koji je garantovao donosiocu novčanice da će dobiti količinu zlata istaknutu na istoj.

Shodno zlatnim rezervama, države su štampale novčanice. Uzimajući u obzir da su šanse da sve novčanice budu zamenjene za zlato u isto vreme izuzetno male, drzave su često štampale više novca nego što imaju rezervi. To nije predstavljalo problem jer je odnos bio uglavnom konstantan što je i omogućavalo stabilost valuta, tj. stabilnost medjusobnih odnosa izmedju valuta. Problem počinje sa Prvim Svetskim ratom gde su države taj odnos ozbiljno poremetile štampajuci sve veće količne novca stvarajući inflaciju valuta i time finansirajući rat. Posle rata zemlje koje su profitirale od rata su se vratile na stari sistem i odnos količine novca i zlata ali na globalnom tržistu se sve manje trgovalo u zlatu, koje je u to doba više korišteno za ratne reparacije.(Warwick-Ching, 1993)

Drugi svetski rat je doveo do jos veće inflacije cena i posle rata je većina valuta prešla na Bretonvudski sporazum gde se više razmena valuta nije vršila kroz njihov odnos prema zlatu, nego prema dolaru, što je u to vreme bilo skoro isto jer je dolar imao konstantu vrednost prema zlatu. Ovom sistemu je došao kraj Niksonovim šokom 1971. kad je dolar postao Fiat valuta i njegova vrednost je prestala da bude fiksna u odnosu na zlato, nego je pušteno da tržiste diktira vrednost. Od tog trenutka predikcija cene zlata postaje zanimljiva tematika. Iako je većina valuta u svetu fiat, centralne banke i dalje drže zlatne rezerve radi likvidnosti. Takodje, u kriznim situacijama kao što je situacija sa pandemijom COVID-19 virusa ljudi pokazuju da imaju veće poverenje u zlato kao čuvara vrednosti nego u valute.(Lewis, 1976)



Slika 4. istorijski prikaz cene zlata u dolarima. Crvena linija predstavlja Niksonov šok avgusta 1971 (macrotrends, 2021)

## 3.2. Prethodna istraživanja predkcije zlata

Potreba za predikcijom cene zlata je nastala od kad je američki dolar postao fiat moneta.

Za predikcije velike tačnosti je bilo dovoljno raditi lineranu regresiju jer je samo tržiste bilo vrlo stabilno. Ali usled pandemije virusa COVID-19 ljudi su pokazali da je poverenje i dalje u zlatu, što se može videti naglim skokom cene usled povećane potražnje. Pošto je došlo do nestabilnosti na tržistu, modeli linearne regresije više ne prate trend dovoljno dobro, a u vreme nestabilnosti dobar model je bitniji nego ikad jer nestabilnost predstavlja šansu za zaradu trgovanjem. Usled pomenute potrebe različiti istraživači su isprobavali koji modeli mašinskog učenja najbolje vrše predikciju i ustanovili su da ARIMA model daje najbolje rezultate, ali primenom modela poput Decision tree na kotrljajući prozor se dobijaju još bolji rezultati jer se smanjenjem dataseta smanjuje inertnost modela i preciznost modela dostiže i do 96% u tačnosti predikcije vrednosti za sutrašnji dan. Primer takvog modela je obradjen u master radu Jovane Stokanović Šević, a dodatna poboljšanja su možda moguća ako se podaci o broju novozaraženih i umrlih o koroni dodatno klasifikuju po geografskim regijama, jer je pretpostavka da situacija pandemije u razlicitim delovima sveta, za početak po kontinentima, drugačije utiče na kretanje cene zlata.(Jovana Stokanović Šević, 2020)

Tabela 1. Prikaz prethodnih istraživanja zlata kao finansijskog sredstva i modela mašinskog učenja.

|  |  |  |  |
| --- | --- | --- | --- |
| **Naziv autora** | Naziv rada | Opis istraživanja | Rezultat istraživanja |
| (Azzutti, A., 2016) | Forecasting Gold Price  A Comparative Study | Evaluacija različitih tehnika za predviđanje cene zlata | Iako različiti modeli daju dobre rezultate ARIMA model daje najpreciznije rezultate |
| (Brownlee, 2016) | How To Backtest Machine Learning Models for Time Series Forecasting | Primer kako se može raditi sa vremenski zavisnim podacima | Hronoloski sled podataka, prilikom predikcija vrednosti na osnovu istorijskih podataka je vrlo bitan |
| (Brownlee, 2017) | How to Create an ARIMA Model for Time Series Forecasting in Python | Prikaz ARIMA modela i način kretanja kroz vremenski zavisan dataset | Pokazano je da je u redu koristiti klizni dataset za predikciju. |
| (Riazuddin, 2020) | Predicting Gold Prices Using Machine Learning | Komparacija modela na istom datasetu za predikciju dobiti za 22 dana | Iz ovog rada se može zaključiti da je veoma bitan odabir dataseta i da parametri nevezani za zlato mogu pomoći u boljoj predikciji |
| (Jansen, 2020) | MACHINE LEARNING FOR ALGORITHM TRADING : Master as a PRO applied artificial intelligence and Python for predict systematic strategies for options and stocks. Learn data-driven finance using keras | Knjiga o primeni mašinskog učenja u trgovanju finansijskim istrumentima |  |
| (Warwick-Ching, 1993) | The International Gold Trade | Knjiga o zlatu kao finansijskom instrumentu. |  |
| (Yousef & Shehadeh, 2020) | The Impact of COVID-19 on Gold Price Volatility | Rad o korelaciji između skoka cene zlata i povećanja broja obolelih od virusa | Rad zaključuje da postoji korelacija I da utiče na finansijski sistem |
| (Bingol et.al., 2020) | Gold price prediction in times of financial and geopolitical uncertainty: A machine learning approach | Nalaženje veze između cene zlata i varijabli koje bi mogle predstavljati finansijsku i geopolitičku krizu | ARIMA model najbolje predviđa buduće cene zlata na osnovu varijabli nestabilnosti |
| (Weng et.al., 2020) | Gold price forecasting research based on an improved online extreme learning machine algorithm | Predikcija cene zlata različitim modelima | GA-ROSELM model ina najpreciznije rezultate |
| (Yerra, 2019) | Predicting Tomorrows Gold Price | Predikcija cene zlata je složen problem s obzirom na to da kolebanje cena zlata nije u potpunosti zasnovano na ponudi i potražnji, već zavisi i od mnoštva geopolitičkih i finansijskih faktora. | Kada se vrše predikcije treba voditi računa o odabiru modela, ali i veličini dataseta i hronološkom rasporedu podataka |
| (Altig et.al., 2020) | Economic uncertainty in the wake of the COVID-19 pandemic | Provera parametara nesigurnosti za vreme Covid pandemije | Povećana nesigurnost preduzeća i potrošače generalno čini opreznim za ulaganja |

## 3.3. Korišćene tehnologije

S obzirom da se svaki servis ponaša kao crna kutija može se smatrati da je svaki servis nezavisna aplikacija za sebe. Zbog raznovrsnosti zadataka koje servisi izvršavaju raznovrsne su i tehnologije kojie su primenjene.

Za lokalno pokretanje korišćen je Docker, za servise su korišćeni programski jezici Python i JavaScript. Frontend servis je realizovan uz pomoć React biblioteke. Broker je Node.js sa Express frejmvorkom i socekt.io dodatkom za web-sockete i got paket za pozive ka drugim servisima. Prediktor i Modelator su napravljeni u Flask frejmworku u python programskom jeziku i oba koriste pandas paket za manipulaciju podacima. Prediktor koristi dodatni paket za zakazivenje egzekucije koda, apscheduler, i paket za pravljenje zahteva ka drugim servisima, requests. Za modelator servis karakterističan je scikit-learn koji je moćan alat za mašinsko učenje.

#### 3.3.1. Docker

Za pokretanje aplikacije na lokalu je korišćen Docker i Docker-Compose. Docker i DockerCompose su odabrani jer je to jedan od najlakših načina da se pokrene aplikacija na lokalu, a pogotovo kad je aplikacija multi servisna. Docker predstavlja set alata za pokretanje softvera zapakovanih u kontejnere koji se pokreću u virtualnim okruženjima koja su u nivou operativnog sisitema. Pošto kontejneri koriste servise kernela, nema potrebe za dodatnim alociranjem resursa, stoga je ovaj pristup lokalnog deploymenta značajno lakši i efikasniji od standarnih rešenja sa virtualnim mašinama poput vagranta. (wikipedija, iz prvog pasusa)

Svaki servis je zapakovan u svoj kontejner tako da je moguće raditi na pojedinačnim servisima. Pošto je u pitanju više servisni-kontejnerski sistem, za lokalni depoloj je korišćen docker-compose alat. Docker-compose za instrukcije deploya koristi YAML fajl u kome su definisani koji su kontejneri potrebni, da li je potrebno predpakovanje kontejnera pre pokretanja, koji su medjusobni odnosi kontejnera i kako komuniciraju sa spoljnim svetom. (*Docker Frequently Asked Questions (FAQ)*, 2021)

Još jedan od razloga zasto je Docker odabran je zato što radi na svim popularnim operativnim sistemima, tako da nema poblema u kompatibilnosti i takozvanog „Kod mene je radilo“ problema. (*How Docker containers solve No more works on my machine*, 2020)

#### 3.3.2. React

React je frontend biblioteka za manipulaciju virtualnog DOM-a (Document Object Model). Veoma je pogodna za pravljenje single-page frontend aplikacija. Pod single page se smatra da kad se jednom stranica učita sav sadržaj stranice se programski menja iz JavaSkripta. Ovakav pristup je odličan za kompleksne korisničke interfejse. Početno učitavanje stranice zavisi od količine koda, ali pošto je moguće koristiti isti element više puta pogotovo kod stranica sa repetitivnim sadržajem je brže učitati šablon i primeniti ga kroz petlju umesto da se celokupan sadržaj stranice šalje sa servera. Mana ovakvog pristupa je u tome što je u pitanju jedna stranica, što otežava vidljivost u pretragama. Ovaj problem single page aplikacije web pretraživači počinju da rešavaju jer je sve više i više single page aplikacija

Popularnost Reacta je zasnovana na tome što je vrlo lagana biblioteka u smislu da veličine i brzine za razliku od drugih popularnih frejmworka poput Angulara i Vue.js. Osim veličine i brzine bitna je i stabilnost same biblioteke, o čemu govori broj korisnika medju kojima su i velike firme poput facebook-a koji je i kreirao i održava biblioteku.(*React – A JavaScript Library for Building User Interfaces*, 2021)

React takođe ima pandan za kreiranje cross platform aplikacija React-native tako da je moguće uz manje promene koda web aplikaciju pretvoriti u mobile aplikaciju za Android i/ili iOS. (*React Native · Learn Once, Write Anywhere*, 2021)

#### 3.3.3. Express

Express je frejmwork za Node.js koji je specijalizovan za kreiranje web aplikacija, ali je poznat kao server frejmwork za Node.js. Ovakav opis se savršeno uklapa u opis Brokera. Prednost Express-a za Broker servis je u tome što se zahtevi obradjuju asinhrono tako da ne dolazi do zagušenja prilikom čekanja na odgovor, a pritom se sve izvršava u jednoj niti, tako da ne dolazi do zagušenja na procesoru prilikom preskakanja sa niti na nit. (*Express - Node.Js Web Application Framework*, 2021)

Generalna mana Node.js-a, a samim tim i Expressa, je da je izuzetno spor na aritmetičkim operacijama i interakcijama sa relacionim bazama. Medjutim, pošto Broker servis u suštini samo preusmerava zahteve na ostale servise ova mana ne predstavlja problem. (*Running CPU Intensive Task in Nodejs*, 2020)

#### 3.3.4. Socket.io

Socet.io je biblioteka za Node.js koja omogućava dvosmernu komunikaciju izmedju klijenta i servera u realnom vremenu upotrebom WebSocket protokola. Kao i sam Node,js biblioteka je asinhrona i vođena događajima. Primenom ovakve konekcije je moguće da se obveštenja i informacije šalju klijentu tu renutka kad su informacije dostupne. Ovime se realizuju notifikacije u realnom vremenu, a može se i realizovati čet aplikacija. (Arrachequesne, 2021)

Ova biblioteka je dodata radi budućeg razvoja aplikacije. Pošto Modelator servis radi isključivo sa modelima mašinskog učenja, za odredjene kalkulacije i kalibracije je potreban duži vremenski period. Tako da ako klijent pokrene dugotrajnu predikciju može doći do odbijanja konekcije usled predugačkog vremena za odgovor na zahtev. Ovaj problem se može rešiti websocket protokolom. Kad klijent posalje zahtev dobija odogvor da je kalkulacija započeta, a onda se rezultat šalje klijentu web socketom na dogadjaj završetka proračuna ili ako klijent nije dostupan rezultat se čuva uz nalog klijenta i onda se na dogadjaj konekcije na server šalju informacije.

#### 3.3.5. Got

Kad expres primi zahtev taj zahtev se dalje prosleđuje odgovarajućem servisu tako što sam broker šalje novi zahtev servisu. Inicijalni plan je bio da se koristi request biblioteka, ali tu biblioteku više niko ne održava, pa samim tim predstavlja potencijalan bezbednosni problem u budućnosti. Alternative request biblioteci su got, node-fetch, axios(iskorišćen unutar frontend servisa) i super agent. Got je odabran kao najbogatija biblioteka koja ima i mogućnosti koje prevazilaze mogućnosti request biblioteke. Te dodatne mogućnosti se trenutno ne koriste ali bi mogle biti značajne u budućnosti. Neke od dodatnih mogućnosti su ponovno slanje zahteva u slušaju greške prilikom inicijalnog zahteva. Ova mogućnost doprinosi stabilnosti sistema u slučaju da je otežana komunikacija izmedju servisa, ova mogućnost nije dostupna u axios biblioteci koja je korišćena u frontendu. Takođe jedna od korisnih mogućnosti je prihvatanje grešaka sa meta podacima što moze biti korisno u rešavanju bagova u toku rada.(Liz Parody, 2019; Sorhus, 2014/2021)

#### 3.3.6. Flask

Flask je mikro framwork za python. Mikro frejmwork je zato što ne zavisi od dodatnih biblioteka i zato što nema dodatnih mogućnosti koje inače većina frejmworka ima, poput autorizacije i sloja za apsrakciju baze podataka. Sve dodatne mogućnosti su dostupne u vidu plugina ili se mogu ručno realizovati u skaldu sa potrebama aplikacije.(*Foreword — Flask Documentation (1.1.x)*, 2021)

Ovakva minimalnost frejmworka je idealna za servise jer nema nepotrebnog opterećenja i zbog toga se koristi u dva servisa aplikacije ovog master rada, Predictor i Modelator.

Ovaj frejmwork se takođe ponaša kao server, slično kao i Express za Node.js tako da se putanje direktno definišu u kodu i nema potrebe za dodatnim slojevima za realizaciju servisa.

Flask je odabran za servise modelator i prediktor zbog svoje jednostavnosti i slobode koju pruža, tako da su servisi realizovani u funkcionalnim programiranjem.

#### 3.3.7. Apscheduler

Za realizaciju Predictor servisa se pojavila potreba za zakazivanjem zadataka radi dobijanja najsvežije predikcije. Da je implementaicja servisa na realnoj mašini bilo bi lako podesti izvršavanje zadataka uz pomoc cron servisa, ali pošto se lokalni deploy radi kroz docker compose, pitanje je kako bi se svaki put podešavao cron. Tako da je odlučeno da se ovom problemu pristupi kroz sam servis (kod).

Zadati se mogu zakazivati tako da se izvrše samo jednom ili da se izvršavaju periodično ili da se izvršavaju na zadate datume. Prednosti ovakovog pristupa u odnosu na cron servis su pristupačnost, razmena i čuvanje podešavanja i nezavisnost od platforme. Pod pristupačnosti se misli na to što je moguće podesiti zadatke direktno u kodu ili čak napraviti interfejs za zakazivanje zadataka sa daljine. Mogućnost čuvanja dogadjaja u bazi omogućava da se ne pogube prilikom restartovanja servisa, a omogućava i deljenje zadataka prilikom promene sistema koji poziva zadatke. Nezavisnost od platforme je možda najbitnij aspekt ove biblioteke jer omogućava da servis može biti implementiran na bilo kojoj platformi koja podržava python i ima pristup mreži. (*Advanced Python Scheduler — APScheduler 3.7.0 documentation*, 2021)

#### 3.3.8. Pandas

Pandas je biblioteka koja omogućava manipulaciju i analizu podataka u pythonu. Biblioteka omogućava strukture za čuvanje i manupulacije serija podataka poput tabela i podataka sa vremenskom komponentom.

Glavni objekat koji pruža biblioteka i nad kojim se vrši manupulacija se zove dataframe. Pandas biblioteka ima mogućnost da se podaci učitavaju u dataframe objekte iz raznovrsnih izvora poput: csv fajlova, eksel fajlova, JSON stringova, ili sql-om iz različitih baza.

Pandas dozvoljava različite operacije nad podacima poput spajanja, pretrage, prečišćavanja i konverzije iz sirovih podataka. (McKinney, 2011)

Pandas biblioteka je korišćena zbog brzine i jednostavnosti rada sa podacima i jer Scikit-learn biblioteka u svojim operacijama barata sa dataframe objektima.

#### 3.3.9. Scikit-learn

Sickit-learn je python biblioteka za mašinsko učenje. Ona u sebi sadrži različite algoritme za klasifikacije, regresije i klasterovanje. U konkretnom slučaju predikcije se koristi decisiont tree regresor, a u modelatoru su implementirani još sledeći regresori: linerni regresor, knn, random forest i SVM (support vector machine) pored već pomenutog decision tree algoritma. (Pedregosa et.al., 2011)

Ova biblioteka otvara mogućnosti mašinskog učenja većem auditorijumu nego što je ranije bilo dostupno. Iako u velikom olakšava upotrebu mašinskog učenja i dalje je potrebno znanje programiranja u pythonu što otežava primenu mašinskog učenja u već postojećim sistemima i zahteva izdavajanje vremena i truda da se nauči dodatni programski jezik, za šta mnogi nemaju vremena i kapaciteta. Baš je ovaj problem je jedan od glavnih motiva za izradu aplikacije ovog master rada.

## 3.4. Korišćeni servisi

U duhu arhitekture aplikacija se sastoji iz nekoliko servisa koji su zaduženi za određene delove poslovne logike. Aplikacija ima servise za sve tri uloge koje mogu postojati u servisno orijentisanoj arhitekturi. Postoje dva provajdera servisa Prediktor i Modelator, sa tim što Prediktor igra i ulogu potraživača servisa i to Modelator servisa. Drugi potraživač je Frontend servis, u pitanju je statički servis koji klijentu vraća frontend aplikaciju u React-u prilikom učitavanja aplikacije servis sa klijentske mašine potražuje rezultate od prediktora da bi ih prikazao. I jedan servis da ih sve okupi i u aplikaciju spoji, Broker. Sva komunikacija između servisa se vrši preko brokera. Iako bi servisi mogli međusobno da se povezuju odlučeno je da se to ne radi jer je na ovaj način potrebno buduće validacije staviti samo na jedno mesto, a i lakše se prati medjusobna interakcija servisa i sprečava najveća boljka servisno orijentisanih aplikacija, pravljenje gomile servisa za koje niko nije baš siguran šta i sa čime rade.

Ova četiri servisa i njihovi međusobni odnosi predstavljaju servisno orijentisanu aplikaciju za predvidjanje cene zlata.

#### 3.4.1. Broker

Borker je servis preko kog idu svi zahtevi ka drugim servisima. Ovaj servis je realizovan u Node.js uz upotrebu Express frejmvorka i Socket.io biblioteke i Got biblioteke. Sam servis se sastoji iz dva dela, prijemni i otpremni deo. Prijemni deo se sastoji od dva paralelna interfejsa. Deo za REST zahteve je realizovn u Express-u, a deo za Web Soket je realizovan uz pomoć Socket.io biblioteke. Ova dva dela potom u skladu sa zahtevima koje dobijaju pozivaju odgovajarujći interfejs ka zahtevanom servisu. Interfejsi za pozivanje servisa se nalaze u zasebim fajlovima, tako da svaki servis ima svoj fajl za interakciju sa njim. Ovako izdvojeni, fajlovi za interakciju sa servisima omogućavaju uvodjenje generisanja koda na osnovu specifikacije API-ja servisa koji se dodaje brokeru. Za opšte provere zahteva proveru treba dodati u Prijemni deo Brokera (validacija tokena, provera IP adresa potraživača…) a unutar Otpremnog dela se mogu staviti dodatne provere specifične za same servise (nedovoljan broj parametara za pozivanje servisa…).

Ovaj servis je prilikom lokalnog deploymenta preko docker-a dostupan na adresi [http://localhost:8000](http://localhost:8000/) ako se zatraži GET metodom bez dodatnih parametara Broker će vratiti Frontend servis.

Prijemnik

Otpremnik

Express

Socketio

Frontend

Prediktor

Modelator

Broker

Slika 5. Unutrašnja strukture Broker servisa

#### 3.4.2. Frontend

Frontend je servis koji predstavlja korisnički interfejs. Korisnički interfejs je realizovan kao single page aplikacjia realizovana sa React bibliotekom. Prilikom učitavanja stranice klijent šalje zahtev brokeru da dohvati informacije od prediktora o predvidjenim cenama zlata za idući berzanski dan. Ovaj zahtev se realizuje uz pomoć axios biblioteke. Ovaj servis prati razvoj ostalih servisa jer on predstavlja prozor za iterakciju klijentima, ali nije nužan za rad same aplikacije jer je moguće koristiti aplikaciju direktno preko zahteva što je naročito korisno ako se aplikacija koristi kao dodatni servis za već postojeća rešenja.

Na lokalnom deplojmentu ovaj servis je dostupan na adresi [http://localhost:3000](http://localhost:3000/) ili preko brokera GET metodom bez dodatnih parametara.

#### 3.4.3. Prediktor

Prediktor je servis koji priprema sve što je potrebno za samu predikciju cene zlata. Prilikom prvog pokretanja pokreće se Apscheduler da na odredjeni interval poziva funkciju za pravljenje predikcije i da rezultate te funkcije upiše u globalnu varijablu koja sadrži kompletan dataframe predikcije. Na svaki interval funkcija za predikciju će biti ponovo pozvana i novi rezultati će pregaziti stare rezultate.

Funkcija za predikciju poziva biblioteku yahoo finance odakle dobija dataframe sa informacijama o ceni zalata za prethodnih godinu dana. Tom datafrejmu potom dodaje jos jednu kolonu i u nju upisuje vrednosti Close cene zlata od sutrašnjeg dana, ova vrednost će nam predstavljati y dataset prilikom treniranja modela.

Takav dataset se potom pretvara u csv i zajedno sa dodatnim informacijama o kolonama koje treba izostaviti iz predikcije, kolonama koje predstavljaju y data set i metodu koji treba da se primeni na datasetu se šalje modelatoru na obradu. Kad modelator vrati rezultat, taj rezultat se pretvara u dataframe i vraća se u globalnu vrijablu dataseta.

Potražnja podataka se vrši kroz drugi deo aplikacije koja je realizovana kroz flask mikro frejmwork.

Kad prediktor servis dobije zahtev za informacijama vraća se određeni deo datafrejma iz globalne varijable koji se potom pretvara u json objekat radi lakše manipulacije od strane korisnika.

#### 3.4.4. Modelator

Modelator predstavlja srce aplikacije i odgovor na problem pristupačnosti mašinskom učenju. Modelator je takodje napravljen u flask mikro frejmworku i shodno zahevu poziva odredjeni model. Ceo servis je napravljen da se ponaša kao poziv funkciji što omogućava da se servis hostuje kao serverless funkcija. Ovakav vid hostinga predstavlja dobar izbor jer Modelator nije pod stalnim opterećenjem, a kad jeste, veći resursi mogu pomoći značajno. Ako se hosting plaća samo kad se koristi time se potencijalno smanjuju troškovi jer nam ne treba snažan server koji je uvek dostupan, a potencijalno dobar deo vremena ne radi ništa.

Modelator na raspolaganje daje pet regresivnih alogritama za predikciju mašinskim učenjem:

* Decision Tree
* Random Forest
* Linear regresor
* KNN
* SVM

Ovih pet algoritama su dostupni sa standardnom obukom sa podelom datafrejmana 80-20% delove za train i test dataset i ovaj odnos je moguće definisati prilikom potraživanja servisa. Ovakva predikcija je često korišćena u radovima da se pokaze da mašinsko učenje može da radi predikciju cene zlata, takav pristup ima velikih propusta, jer pogađa nasumičnih 20% dataseta i u obzir uzima skoro ceo dataset gde u suštini pokušava samo da što bolje zakrpi 20% „rupa“ u podacima.

Zbog prethodno navedenih problema modelator ima i mogućnost rolling window predikcije gde je moguće definisati veličinu prozora koji treba uzeti u obzir za treniranje modela pre nego što se testira na prvom sledećem unosu. Ovakav pristup se pokazao vrlo uspešnim u istraživanju Jovane Stokanović Šević.

## 3.5. Arhitektura

Sa pokretanjem instance aplikacije prvu operaciju započinje Prediktor pozivanjem Modelator servisa preko Brokera da bi dobio prvobitnu predikciju cene zlata. Kad prediktor dobije odgovor od Modelatora dataset predikcija se čuva unutar Prediktora.

Odlaskom na naslovnu stranicu aplikacije zahtev dolazi do Brokera koji vraća Frontend servis. Trenutno je moguće tražiti frontend servis direktno jer web socket nije implementiran do kraja, tako da nema potrebe za vodjenjem evidencije konekija. Kad klijent dobije Frontend servis servis se pokreće u okviru pregledača, sa pokretanjem frontend servisa poziva se broker da pozove Prediktor da vrati vrednost predikcije za sledeći investicioni dan, kad Prediktor vrati odgovor i kad odgovor dodje do Frontend servisa, menja se stanje Frontend servisa i prikazuju se informacije klijentu.

Prikaz rezultata

Predikcija

Tok izvršavanja

Vreme Frontend Broker Prediktor Modelator

0

Slika 6. Prikaz toka aplikacije

Osim komunikacije preko korisničkog interfejsa moguće je direktno pristupiti brokeru za ostale servise, ovakvim pristupom je moguće pozivati direktno modelator što predstavlja zanimljivu uslugu za integraciju mašinskog učenja u projekte koji ne mogu lako da inkorporiraju python.

Po ulogama koje servisi imaju dele se u tri grupe servisa: Provajder, Broker i Potraživač. U aplikaciji postoje sevisi za svaku od uloga, a jedan od servisa ima i dvojnu ulogu tako da je i potraživač i povajder.

Broker

Provajder

Potraživač

Slika 7. Prikaz uloga servisa

# 4. Zaključak

Ovom aplikacijom su približene mogućnosti mašinskog učenja široj javnosti. Aplikacija svojom servisno orijentisanom arhitekturom rešava dva problema.

Prvi problem je problem pristupačnosti mašinskog učenja na već postojećim sistemima i platformama koje ne podržavaju komplekse procese poput mašinskog učenja. Drugi problem je predikcija finansijskih derivata poput zlata. Aplikaicja rešava prvi problem servisom za modeliranje, Modleator, u kome je primenjena scikit-learn biblioteka koja sadrži različite algoritme mašinskog učenja. Problem predikcije finansijskih derivata je rešen prediktor servisom koji prikuplja podatke i prirpema ih za predikciju, i nakon uspešne predikcije čuva podatke i omogućava ih dostupnim korisnicima servisa. Prediktor servis se može posmatrati i kao demonstracija potencijala Modelator servisa

Aplikacija ima potencijala za dalji razvoj, a predstavlja i dobru početnu tačku za buduće projekte.

## 4.1. Budući razvoj

Prateći poslednji postulat servisno orijentisane arhitekture koji kaže da je evolutivno usavršavanje bitnije od inicijalne savršenosti u ovom poglavlju se navode smernice za dalje usavršavanje ove aplikacije

Prvo, i možda najznačajnije unapredjenje ove aplikacije bi bilo uvođenje User servisa. Ovaj servis bi bio zadužen za kontrolu pristupa aplikaciji. Ovaj servis bi bio od ključnog značaja ako bi aplikacija prešla u produkcijsku fazu, jer je modelator servis jako lako preopteretiti do neupotrebljivosti.

Sladeći korak u razvoju bi trebalo da bude servis za čuvanje rezultata modela klijenata. Sa ovim servisom bi implementirana web socket tehnologija mogla da pokaže svoj pun potencijal tako što klijent dobije rezultate predikcija koje su se izračunale dok je kljient bio odsutan. Predlaže se da se servis za čuvanje odvoji od User servisa da ne bi došlo do preopterećenja user servisa i da se smanji mogućnost za pravljenje grešaka unutar User servisa što bi moglo da dovede do ozbiljnih bezbednosnih problema.

Kad su ova dva produkcijski nepohodna servisa dodata sledeći koraci se mogu dodavati bilo kojim redom.

Jedan od najlakših koraka bi bio da se Modelator servisu doda mogućnost primene i ostalih algoritama tako da bi zaista predstavljao jedan potpun servis za mašinsko učenje.

Osim modelatora Prediktoru bi mogle da se dodaju nove mogućnosti poput različitih finansijskih derivata, čuvanja rezultata u bazi, računanje tačnosti predikcija, predikcije za više od jednog dana u napred i predikcija celog dataseta samo ako dataset nema već predikciju, a ako ima da se radi samo predikcija za dane koji nemaju predikciju. Sa ovim unapredjenjima Prediktor servis bi bio značajan finansijski/investicioni alat.

Unapredjenja Modelatora i Prediktora bi trebalo da prate i izmene u Frontend servisu jer su sve izmene nebitne ako nisu primenljive. Frontend servis posle izmena na modelatoru i prediktoru mogao bi da se podeli na dva dela, Forntend za predkiciju finansijskih derivata i Frontend za mašinsko učenje i time bi ova aplikacija mogla da se predstavi kao dve odvojene aplikacije iako koriste iste servise.

Broker unapredjenja GraphQL, generijsanje koda za konekcije

Ova aplikaicja predstavlja dobru odskočnu dasku za druge aplikacije jer se funkionalnost može u potpunosti promeniti izmenom samo par servisa. Čak bi i bilo moguće dodati još servisa uz već postojeće i onda samo user interfejsom (frontendom) izdvojiti različite aplikacije iako u pozadini dele iste servise.

Konkretno ova aplikacija ima dosta prostora za usavršavanje funkcionalnosti. Unutar prediktor servisa može se dodati jos finansijskih derivata poput investicionog srebra i akcija sa berze. Takodje se moze dati da ako klijent traži neku vrednost koja ranije nije bila korišćena da se uradi predikcija za tu vrednost i da se rezultati sačuvaju tako da budu dostupni i ostalim klijentima. Može se dodati da se na update predikcije šalje zahtev brokeru da obavesti frontend svih klijenata preko web socketa, iz tog razloga broker ima socket.io biblioteku primenjenu. Modelator servisu dodati mogućnost slanja dodatnih opcija koje modeli pirhvataju, tako da je moguće još detaljnije podesiti same modele. Modelator je takodje dobar kandidat za serverless deployment jer se ponaša kao funkcija koja uvek vraća isti rezultat za iste ulazne podatke, takođe moguće je da je nekim modelima potrebno dosta vremena da se izvrše što bi moglo da dovede do pretrpavanja klasičnih deployment rešenja, a sa serverless pristupom se naplaćuje samo vreme egzekucije i moguće je da neograničeno traje. Ako se doda jos User servis moguće je i naplaćivati korisnicima upotrebu aplikacije, tako da bi se i sami troškovi serverles izvršavanja odmah prebacivali na krajnje korisnike. Frontend bi trebalo da prati sva pomenuta dalja usavršavanja, tako da je potrebno napraviti i interfejs za direktnu upotrebu modela, i interfejs za odabir i dodavanje finansijskih derivata.

# 5. Literatura

1. *Advanced Python Scheduler—APScheduler 3.7.0 documentation*. (2021). https://apscheduler.readthedocs.io/en/stable/
2. Altig, D. E., Baker, S., Barrero, J. M., Bloom, N., Bunn, P., Chen, S., Davis, S., Leather, J., Meyer, B., Mihaylov, E., Mizen, P., Parker, N., Renault, T., Smietanka, P., & Thwaites, G. (2020, Јули 24). Economic uncertainty in the wake of the COVID-19 pandemic. *VoxEU.org*. https://voxeu.org/article/economic-uncertainty-wake-covid-19-pandemic
3. Arrachequesne, D. (2021, Април 11). *Socket.IO*. Socket.IO. https://socket.io/index.html
4. Azzutti, A. (2016). *Forecasting Gold Price A Comapative Study*.
5. Bell, M. (2008). *Service-oriented modeling: Service analysis, design, and architecture*. Hoboken, N.J. : John Wiley & Sons. http://archive.org/details/serviceorientedm00bell
6. Bingol, S., Sadik, S., Abu Bakar, R. M., Mouhcine, H., Albaf, J., Chaabene, S., & Auta, A. (2020). *Gold price prediction in times of financial and geopolitical uncertainty: A machine learning approach*. https://doi.org/10.13140/RG.2.2.31050.39366
7. Brownlee, J. (2016, Децембар 18). How To Backtest Machine Learning Models for Time Series Forecasting. *Machine Learning Mastery*. https://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/
8. Brownlee, J. (2017, Јануар 8). How to Create an ARIMA Model for Time Series Forecasting in Python. *Machine Learning Mastery*. https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/
9. Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., & Stafford, J. (2010). *Documenting Software Architectures: Views and Beyond* (2nd edition). Addison-Wesley Professional.
10. DA-14. (2018, Фебруар 7). *The Ultimate Guide to API Architecture: REST, SOAP or GraphQL? | DA-14*. https://da-14.com/blog/ultimate-guide-api-architecture-rest-soap-or-graphql
11. *Docker frequently asked questions (FAQ)*. (2021, Април 14). Docker Documentation. https://docs.docker.com/engine/faq/
12. Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, Today, and Tomorrow. У M. Mazzara & B. Meyer (Ур.), *Present and Ulterior Software Engineering* (стр. 195–216). Springer International Publishing. https://doi.org/10.1007/978-3-319-67425-4\_12
13. *Express—Node.js web application framework*. (2021). https://expressjs.com/
14. *Foreword—Flask Documentation (1.1.x)*. (2021). https://flask.palletsprojects.com/en/1.1.x/foreword/
15. Garlan, D., & Shaw, M. (1993). AN INTRODUCTION TO SOFTWARE ARCHITECTURE. У V. Ambriola & G. Tortora, *Series on Software Engineering and Knowledge Engineering* (Том 2, стр. 1–39). WORLD SCIENTIFIC. https://doi.org/10.1142/9789812798039\_0001
16. *How Docker containers solve No more works on my machine*. (2020, Април 15). Stack Overflow. https://stackoverflow.com/questions/61220475/how-docker-containers-solve-no-more-works-on-my-machine
17. Jansen, S. (2020). *Machine Learning for Algorithmic Trading: Predictive models to extract signals from market and alternative data for systematic trading strategies with Python, 2nd Edition*. Packt Publishing.
18. Jovana Stokanović Šević. (2020). *Model predikcije kretanja cena zlata uzimajući u obzir broj zaraženih od Covid-19 virusa*. Singidunum University.
19. Lange, K. (2016). *The Little Book on REST Services*. 31.
20. Laws, M. P., & Goodfellow, M. G. (2018). *Choosing a Web Host: How to Choose the Web Hosting Service that is Best for You* (1st edition). CreateSpace Independent Publishing Platform.
21. Leach, P. J., Berners-Lee, T., Mogul, J. C., Masinter, L., Fielding, R. T., & Gettys, J. (1999). *Hypertext Transfer Protocol—HTTP/1.1*. https://tools.ietf.org/html/rfc2616
22. Lewis, P. (1976, Август 15). Nixon’s Economic Policies Return to Haunt the G. O. P. *The New York Times*. https://www.nytimes.com/1976/08/15/archives/nixons-economic-policies-return-to-haunt-the-gop-nixons-economic.html
23. Liz Parody. (2019, Април 3). *Request is going into maintenance mode, this is what you need to know*. The NodeSource Blog - Node.Js Tutorials, Guides, and Updates. http://nodesource.com/blog/express-going-into-maintenance-mode/
24. macrotrends. (2021, Април 14). *Historical Gold price graph*. https://www.macrotrends.net/1333/historical-gold-prices-100-year-chart
25. McKinney, W. (2011). *pandas: A Foundational Python Library for Data Analysis and Statistics*. 9.
26. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*(85), 2825–2830.
27. Peter Lubbers & Frank Greco. (2020). *HTML5 WebSocket—A Quantum Leap in Scalability for the Web*. http://www.websocket.org/quantum.html
28. Pressman, R. S., & Maxim, B. (2014). *Software Engineering: A Practitioner’s Approach* (8th edition). McGraw-Hill Education.
29. *React – A JavaScript library for building user interfaces*. (2021). https://reactjs.org/
30. *React Native · Learn once, write anywhere*. (2021). https://reactnative.dev/
31. Riazuddin, M. (2020, Април 20). *Machine Learning to Predict Gold Price Returns*. Medium. https://towardsdatascience.com/machine-learning-to-predict-gold-price-returns-4bdb0506b132
32. Rotem-Gal-Oz, A. (2012). *SOA Patterns* (1st edition). Manning Publications.
33. *Running CPU Intensive task in Nodejs*. (2020, Април 20). DEV Community. https://dev.to/kayslaycode/running-cpu-intensive-task-in-nodejs-201k
34. *SOA Manifesto*. (2009). http://www.soa-manifesto.org/
35. Sorhus, S. (2021). *Sindresorhus/got* [TypeScript]. https://github.com/sindresorhus/got (Original work published 2014)
36. Warwick-Ching, T. (1993). *The International Gold Trade*. Woodhead Publishing.
37. Weng, F., Yinghao, C., Wang, Z., Hou, M., Luo, J., & Tian, Z. (2020). Gold price forecasting research based on an improved online extreme learning machine algorithm. *Journal of Ambient Intelligence and Humanized Computing*, *11*. https://doi.org/10.1007/s12652-020-01682-z
38. Yerra, B. (2019, Октобар 12). *Predicting Tomorrows Gold Price*. https://mlbhanuyerra.github.io/2019-10-12-Predicting-Tomorrows-Gold-Price/
39. Yousef, I., & Shehadeh, E. (2020). The Impact of COVID-19 on Gold Price Volatility. У *International Journal of Economics and Business Administration: Том VIII*. https://doi.org/10.35808/ijeba/592

# 6. Prilozi

## 6.1. Github

Kod projekta je dostupan na repozitorijumu <https://github.com/Msevic/master>.

## 6.2. Struktura fajlova

|  |  |
| --- | --- |
|  |  |
| Struktura celog projekta | Struktura Broker servisa |
|  |  |
| Struktura Frontend servisa | Struktura Modelator servisa |
|  |  |
| Struktura Prediktor Servisa |  |

## 6.3. Van servisni fajlovi

### 6.3.1. README.md

1. *# Master*
2. Application for predicting gold price
3. Created for the needs of master thesis
4. Use docker compose to run this software

### 6.3.2. LICENSE

1. GNU GENERAL PUBLIC LICENSE
2. Version 3, 29 June 2007

### 6.3.3. docker-compose.yml

1. Version: "3.4"
2. services:
3. broker:
4. build:
5. context: ./broker
6. network: host
7. command: npm run dev
8. volumes:
9. - ./broker:/usr/src/app/
10. ports:
11. - "8000:8000"
12. frontend:
13. build: frontend
14. command: npm start
15. volumes:
16. - ./frontend:/app/
17. ports:
18. - "3000:3000"
19. environment:
20. - CHOKIDAR\_USEPOLLING=true
21. predictor:
22. build:
23. context: ./predictor
24. network: host
25. volumes:
26. - ./predictor/:/usr/src/app/
27. ports:
28. - 5000:5000
29. environment:
30. PORT: 5000
31. FLASK\_DEBUG: 1
32. depends\_on:
33. - modelator\_d3\_rw
34. modelator\_d3\_rw:
35. build:
36. context: ./modelator\_d3\_rw
37. network: host
38. volumes:
39. - ./modelator\_d3\_rw/:/usr/src/app/
40. ports:
41. - 6001:6001
42. environment:
43. PORT: 6001
44. FLASK\_DEBUG: 1

### 6.3.4 .gitignore

1. .idea

## 6.4. Fajlovi Broker Servisa

### 6.4.1. broker/sevices/forntend/service.js

1. *frontend* = async (app) => {
2. app.get('/', function (req, res) {
3. res.redirect('[http://localhost:3000](http://localhost:3000/)')
4. });
5. };
6. module.exports = *frontend*

### 6.4.2. broker/sevices/modelator\_d3\_rw/service.js

1. const got = require('got');
2. const proxy = require('express-http-proxy');
3. *md3rw* = (app) => {
4. app.get('/md3rw', function (req, res) {
5. got('http://modelator\_d3\_rw:6001').then(response => {
6. res.send(response.body);
7. }).catch(error => {
8. res.send(error.response.body);
9. });
10. });
11. app.use('/md3rw/rw/d3', proxy('modelator\_d3\_rw:6001', {
12. reqBodyEncoding: null,
13. preserveHostHdr: true,
14. proxyReqPathResolver: function(req){
15. return "/rw/d3";
16. }
17. }));
18. }
19. module.exports = *md3rw*

### 6.4.3 broker/sevices/predictor/service.js

1. const got = require('got');
2. *predictor* = (app) => {
3. app.get('/predictor', function (req, res) {
4. got('http://predictor:5000/').then(response => {
5. res.send(response.body);
6. }).catch(error => {
7. console.log(error.response.body);
8. });
9. });
10. app.get('/predictor/predictions/:commodity?/:count?', (req, res) => {
11. var url = '<http://predictor:5000/predictions/>';
12. if( req.params.commodity){
13. var count = req.params.count || 0;
14. url += req.params.commodity + "/" +count
15. }
16. got(url).then(response => {
17. res.send(response.body);
18. }).catch(error => {
19. res.send(error.response.body);
20. });
21. })
22. }
23. module.exports = *predictor*

### 6.4.4 broker/index.js

1. var express = require('express');
2. var app = express();
3. var http = require('http').Server(app);
4. var io = require('socket.io')(http);
5. var *frontend* = require('./sevices/forntend/service')
6. var *predictor* = require('./sevices/predictor/service')
7. var *md3rw* = require('./sevices/modelator\_d3\_rw/service')
8. *frontend*(app)
9. *md3rw*(app)
10. app.get('/md3rw', function (req, res) {
11. *md3rw*(res)
12. });
13. *predictor*(app)
14. io.on('connection', function (socket) {
15. console.log("New Connection")
16. });
17. http.listen(8000, function () {
18. console.log('listening on \*:8000');
19. });

### 6.4.5. broker/Dockerfile

1. FROM node:14
2. RUN npm install -g nodemon
3. RUN npm install express
4. RUN npm install socket.io
5. RUN npm install got
6. RUN npm install express-http-proxy
7. *# Create app directory*
8. WORKDIR /usr/src/app
9. *# Install app dependencies*
10. *# A wildcard is used to ensure both package.json AND package-lock.json are copied*
11. *# where available (*[npm@5](mailto:npm@5)*+)*
12. COPY package\*.json ./
13. RUN npm install
14. *# If you are building your code for production*
15. *# RUN npm ci –only=production*
16. EXPOSE 8000
17. CMD [ "node", "server.js" ]

### 6.4.6. broker/package.json

1. {
2. "name": "broker",
3. "version": "1.0.0",
4. "description": "Node.js Broker",
5. "author": "Mihailo Sevic",
6. "main": "index.js",
7. "scripts": {
8. "start": "node index.js",
9. "dev": "nodemon index.js"
10. },
11. "dependencies": {
12. "express": "^4.16.1",
13. "socket.io": "^2.2.0",
14. "nodemon": "^1.18.7"
15. },
16. "devDependencies": {
17. "nodemon": "^1.18.7"
18. }
19. }

## 6.5. Fajlovi Frontend servisa

Fajlovi frontend servisa su autogenersani na osnovu package.json fajla. Jedino su promenjeni App.js, Dockerfile i package.json fajl.

### 6.5.1 frontend/package.json

1. {
2. "name": "frontend",
3. "version": "0.1.0",
4. "private": true,
5. "dependencies": {
6. "axios": "^0.21.1",
7. "react": "^17.0.1",
8. "react-dom": "^17.0.1",
9. "react-scripts": "1.1.5"
10. },
11. "scripts": {
12. "start": "react-scripts start",
13. "build": "react-scripts build",
14. "test": "react-scripts test –env=jsdom",
15. "eject": "react-scripts eject"
16. }
17. }

### 6.5.2. frontend/Dockerfile

1. FROM node:14
2. *# set working directory*
3. WORKDIR /app
4. *# add `/app/node\_modules/.bin` to $PATH*
5. ENV *PATH* /app/node\_modules/.bin:$*PATH*
6. *# install app dependencies*
7. COPY package.json ./
8. COPY package-lock.json ./
9. RUN npm install –silent
10. RUN npm install react-scripts@3.4.1 -g –silent
11. *# add app*
12. COPY . ./
13. *# start app*
14. CMD ["npm", "start"]

### 6.5.3. frontend/src/App.js

1. import React, {Component} from 'react';
2. import logo from './logo.svg';
3. import './App.css';
4. import axios from 'axios'
5. class App extends Component {
6. state = {
7. today: {
8. Date: "",
9. CLose: 0,
10. High: 1,
11. Low: 0,
12. Prediction: 0
13. }
14. }
15. componentDidMount() {
16. axios.get('<http://localhost:8000/predictor/predictions/GLD/1>')
17. .then(response => {
18. var data = response.data
19. var keys = Object.keys(data)
20. var thekey = keys.reduce(function (p, v) {
21. return (p > v ? p : v);
22. });
23. this.setState({
24. today: {
25. Date: response.data[thekey].Date,
26. Close: response.data[thekey].Close,
27. High: response.data[thekey].High,
28. Low: response.data[thekey].Low,
29. Prediction: response.data[thekey].prediction\_d3
30. }
31. })
32. })
33. }
34. render() {
35. return (
36. <div className="App">
37. <header className="App-header">
38. <h1 className="App-title">Predikcija cene zlata za sutrašnji dan</h1>
39. </header>
40. <p className="App-intro">
41. Date : {this.state.today.Date} <br/>
42. High : {this.state.today.High} <br/>
43. <b>Close : {this.state.today.Close}</b> <br/>
44. Low : {this.state.today.Low} <br/>
45. Prediction for next day close : {this.state.today.Prediction} <br/>
46. </p>
47. </div>
48. );
49. }
50. }
51. export default App;

## 6.6 Fajlovi Modelator servisa

### 6.6.1. modelator\_d3\_rw/app.py

1. import os
2. from flask import Flask, jsonify, request
3. from modelator import modelator
4. app = Flask(\_\_name\_\_)
5. @app.route(**'/'**)
6. def welcome():
7. return jsonify({**'status'**: **'modelator working'**})
8. @app.route(**'/rw/<model>'**, methods=[**'POST'**])
9. def rw(model=False):
10. if request.method != **'POST'**:
11. return 404
12. if not model:
13. return jsonify({**'available models'**: [**'d3'**, **'lin\_r'**, **'r\_forest'**, **'knn'**, **'svm\_model'**]})
14. split = True
15. rolling\_window = True
16. *# dataframes*
17. test = False
18. if **'test.csv'** in request.files:
19. split = False
20. test = request.files.get(**'test.csv'**)
21. train = request.files[**'train.csv'**]
22. frame\_size = 10
23. test\_size = 0.8
24. if rolling\_window:
25. frame\_size = int(request.form[**'frame\_size'**])
26. if frame\_size < 3:
27. frame\_size = 10
28. else:
29. test\_size = request.form[**'test\_size'**]
30. target\_column = request.form[**'target\_column'**]
31. return modelator(model, split, rolling\_window, train, test, target\_column, frame\_size, test\_size)
32. if \_\_name\_\_ == **'\_\_main\_\_'**:
33. app.run(host=**'0.0.0.0'**, port=os.getenv(**'PORT'**))

### 6.6.2. modelator\_d3\_rw/modelator.py

1. from rolling\_window import roller
2. from standard\_model import standard\_model
3. def modelator(model, split, rolling\_window, train, test, target\_column, frame\_size=10,

test\_size=0.8):

1. if rolling\_window:
2. return roller(model, train, frame\_size, target\_column)
3. else:
4. return standard\_model(model, train, target\_column, split, test)

### 6.6.3. modelator\_d3\_rw/models.py

1. from sklearn import svm
2. from sklearn.ensemble import RandomForestRegressor
3. from sklearn.neighbors import KneighborsRegressor
4. from sklearn.tree import DecisionTreeRegressor
5. from sklearn.linear\_model import LinearRegression
6. def lin\_r(x\_train, y\_train, x\_test):
7. lin = LinearRegression()
8. lin.fit(x\_train, y\_train)
9. lin\_pred = lin.predict(x\_test)
10. return lin\_pred
11. def r\_forest(x\_train, y\_train, x\_test):
12. rfr = RandomForestRegressor(n\_estimators=100, min\_samples\_leaf=10,

random\_state=1)

1. rfr.fit(x\_train, y\_train)
2. predictions = rfr.predict(x\_test)
3. return predictions
4. def knn(x\_train, y\_train, x\_test):
5. n\_neighbors = 2
6. knn\_m = KNeighborsRegressor(n\_neighbors, weights=**'distance'**)
7. knn\_m.fit(x\_train, y\_train)
8. pred\_knn = knn\_m.predict(x\_test)
9. return pred\_knn
10. def d3(x\_train, y\_train, x\_test):
11. dtr = DecisionTreeRegressor()
12. dtr = dtr.fit(x\_train, y\_train)
13. pred\_dtc = dtr.predict(x\_test)
14. return pred\_dtc
15. def svm\_model(x\_train, y\_train, x\_test):
16. svm\_model = svm.SVR(gamma=**'scale'**)
17. svm\_model = svm\_model.fit(x\_train, y\_train)
18. pred\_svm = svm\_model.predict(x\_test)
19. return pred\_svm

### 6.6.4. modelator\_d3\_rw/responder.py

1. from flask import Response
2. def respond(df):
3. return Response(
4. df.to\_csv(),
5. mimetype=**"text/csv"**,
6. headers={**"Content-disposition"**: **"attachment; filename=y\_test.csv"**, **"Content-Type"**: **"text/csv"**})

### 6.6.5. modelator\_d3\_rw/rolling\_window.py

1. import models
2. from responder import respond
3. import pandas as pd
4. import numpy as np
5. from splitter import splitter\_rolling
6. def roller(model, train, frame\_size, target\_column):
7. train = pd.read\_csv(train, index\_col=0)
8. train = train[~train.isin([np.nan, np.inf, -np.inf]).any(1)]
9. method\_to\_call = getattr(models, model)
10. if method\_to\_call:
11. i = frame\_size
12. exclude\_column = **'prediction\_'** + str(model)
13. exclude\_columns = [exclude\_column, **"Date"**]
14. train[exclude\_column] = 0
15. while i < (len(train.index) – 1):
16. i += 1
17. x\_train, y\_train, x\_test = splitter\_rolling(train, frame\_size, i, target\_column, exclude\_columns)
18. train.loc[[i], **'prediction\_'** + str(model)] = method\_to\_call(x\_train, y\_train, x\_test)[0]
19. return respond(train)
20. else:
21. return jsonify({**'error'**: **'selected model not available'**})

### 6.6.6. modelator\_d3\_rw/splitter.py

1. from sklearn.model\_selection import train\_test\_split
2. def splitter(df, target\_column, test\_size):
3. columns = df.columns.tolist()
4. columns = [c for c in columns if c not in [target\_column]]
5. X = df[columns]
6. y = df[target\_column]
7. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=test\_size, random\_state=123)
8. return X\_train, X\_test, y\_train, y\_test
9. def splitter\_rolling(df, frame\_size, start\_index, target\_column, exclude\_columns):
10. start\_index = start\_index - frame\_size – 1
11. end\_index = start\_index + frame\_size
12. columns = df.columns.tolist()
13. excludeArray = [target\_column] + exclude\_columns
14. columns = [c for c in columns if c not in excludeArray]
15. x = df[columns]
16. y = df[target\_column]
17. *# Delimo podatke na train i test set*
18. x\_test = x.iloc[[end\_index + 1]]
19. x\_train = x.iloc[[start\_index, end\_index]]
20. y\_train = y.iloc[[start\_index, end\_index]]
21. return x\_train, y\_train, x\_test

### 6.6.7. modelator\_d3\_rw/standard\_model.py

1. import pandas as pd
2. import models
3. from responder import respond
4. from splitter import splitter
5. def standard\_model(model, train, target\_column, split, test):
6. train = pd.read\_csv(train)
7. if split:
8. x\_train, x\_test, y\_train, y\_test = splitter(df, target\_column, test\_size)
9. else:
10. test = df.read\_csv(test)
11. y\_train = train[target\_column]
12. x\_train = train.drop(target\_column, axis=1)
13. x\_test = test.drop(target\_column, axis=1)
14. method\_to\_call = getattr(models, model)
15. y\_test = method\_to\_call(x\_train, y\_train, x\_test)
16. return respond(y\_test)

### 6.6.8. requirements.txt

1. Flask
2. pandas
3. numpy
4. sklearn

### 6.6.9. Dockerfile

1. FROM python:3.6
2. RUN mkdir /usr/src/app/
3. COPY . /usr/src/app/
4. WORKDIR /usr/src/app/
5. EXPOSE 6001
6. RUN pip install -r requirements.txt
7. CMD ["python", "app.py"]

## 6.7.Falovi Predictor servisa

### 6.7.1. predictor/app.py

1. import os
2. import time
3. import atexit
4. import vars
5. from flask import Flask, jsonify
6. from apscheduler.schedulers.background import BackgroundScheduler
7. from predictor import make\_predictions, get\_prediction\_for, update\_predictions
8. vars.init()
9. scheduler = BackgroundScheduler()
10. *# comment for production*
11. scheduler.add\_job(func=update\_predictions, trigger=**"interval"**, hours=0.5)
12. *# uncomment for production*
13. *# scheduler.add\_job(func=make\_predictions, trigger="cron", hour="4")*
14. scheduler.start()
15. *# Shut down the scheduler when exiting the app*
16. atexit.register(lambda: scheduler.shutdown())
17. app = Flask(\_\_name\_\_)
18. @app.route(**'/'**)
19. def welcome():
20. *# return a json*
21. return jsonify({**'status'**: **'predictor working'**})
22. @app.route(**'/predictions/<commodity>/<rows>'**, methods=[**'GET'**])
23. def get\_latest\_prediction(commodity, rows = 0):
24. if commodity not in [**'GLD'**]:
25. return jsonify({**'avalable predictions'**: [**'GLD'**]})
26. return get\_prediction\_for(commodity, rows)
27. if \_\_name\_\_ == **'\_\_main\_\_'**:
28. *# define the localhost ip and the port that is going to be used*
29. *# in some future article, we are going to use an env variable instead a hardcoded port*
30. app.run(host=**'0.0.0.0'**, port=os.getenv(**'PORT'**))

### 6.7.2. predictor/predictor.py

1. import pandas as pd
2. import yfinance as yf
3. from datetime import date, timedelta
4. import requests
5. import vars
6. import io
7. def predict(df, target\_column, window):
8. url = **"**<http://broker:8000/md3rw/rw/d3>**"**
9. *# url = "*<http://modelator_d3_rw:6001/rw/d3>*"*
10. params = {**'frame\_size'**: window,
11. **'target\_column'**: target\_column}
12. files = {**'train.csv'**: io.StringIO(df.to\_csv())}
13. response = requests.post(url, data=params, files=files)
14. return response.content.decode(**'utf-8'**)
15. def get\_covid():
16. df = pd.read\_csv(**'**<https://opendata.ecdc.europa.eu/covid19/casedistribution/csv>**'**)
17. df[**'Date'**] = df[**'dateRep'**]
18. df[**'Date'**] = pd.to\_datetime(df.dateRep, format=**'%d/%m/%Y'**)
19. df = df[[**'Date'**, **'deaths'**, **'cases'**]].groupby(**'Date'**).sum()
20. df[**'C\_7'**] = df[**'cases'**].rolling(window=7).mean()
21. df[**'D\_7'**] = df[**'deaths'**].rolling(window=7).mean()
22. return df
23. def get\_gold(commodity, from\_date, to\_date):
24. dst = yf.download(commodity, from\_date, to\_date)
25. dst = dst[[**'Close'**, **'High'**, **'Low'**, **'Volume'**]]
26. dst = dst.dropna()
27. dst[**'S\_3'**] = dst[**'Close'**].rolling(window=3).mean()
28. dst[**'S\_9'**] = dst[**'Close'**].rolling(window=9).mean()
29. dst[**'c\_to\_guess'**] = dst[**'Close'**].shift(-1)
30. return dst
31. def get\_data(commodity, from\_date, to\_date):
32. df = get\_gold(commodity, from\_date, to\_date)
33. *# covid = get\_covid()*
34. *# df = pd.merge(gold, covid, how='left', on='Date').drop\_duplicates()*
35. df.loc[[df.tail(1).index.item()], **'c\_to\_guess'**] = 0
36. df = df.dropna()
37. df[**'Date'**] = df.index
38. df = df.reset\_index(drop=True, inplace=False)
39. return df
40. def make\_predictions(commodity=**'GLD'**):
41. to\_date = date.today().strftime(**"%Y-%m-%d"**)
42. from\_date = **'2020-01-01'**
43. df = get\_data(commodity, from\_date, to\_date)
44. window = 10
45. target\_column = **"c\_to\_guess"**
46. predicted\_df = predict(df, target\_column, window)
47. predicted\_df = io.StringIO(predicted\_df)
48. result = pd.read\_csv(predicted\_df, index\_col=0)
49. return result
50. def get\_prediction\_for(commodity, rows):
51. rsp = getattr(vars, **"df"** + commodity)
52. rows = int(rows)
53. if rows != 0:
54. rsp = rsp.tail(rows)
55. *# return rsp.to\_csv()*
56. return rsp.to\_json(orient=**'index'**)
57. def update\_predictions():
58. vars.update()
59. *# notify frontend with latest data*

### 6.7.3. predictor/vars.py

1. from predictor import make\_predictions
2. def init():
3. global dfGLD
4. dfGLD = make\_predictions()
5. *# dfGLD = 1*
6. def update():
7. global dfGLD
8. dfGLD = make\_predictions()
9. *# dfGLD += 1*

### 6.7.4. predictor/Dockerfile

1. FROM python:3.6
2. RUN mkdir /usr/src/app/
3. COPY . /usr/src/app/
4. WORKDIR /usr/src/app/
5. EXPOSE 5000
6. RUN pip install -r requirements.txt
7. CMD ["python", "-u", "app.py"]

### 6.7.5. predictor/requirements.txt

1. Flaskapscheduler
2. pandas
3. yfinance
4. requests