Važnost, svrhovitost te primjer tehničke izvedbe Laboratory Information Management Systema u biotehnološkim tvrtkama

Bakota, Domagoj

Master's thesis / Diplomski rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Pharmacy and Biochemistry / Sveučilište u Zagrebu, Farmaceutsko-biokemijski fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:163:911708

Domagoj Bakota

Važnost, svrhovitost te primjer tehničke izvedbe Laboratory Information Management Systema u biotehnološkim tvrtkama

DIPLOMSKI RAD

Predan Sveučilištu u Zagrebu Farmaceutsko-biokemijskom fakultetu

Zagreb, 2020.

Ovaj diplomski rad prijavljen je na kolegiju Molekularna biologija s genetičkim inženjerstvom Sveučilišta u Zagrebu Farmaceutsko-biokemijskog fakulteta i izrađen na Zavodu za biokemiju i molekularnu biologiju pod stručnim vodstvom prof.dr.sc. Gordana Lauca

Ovom prilikom zahvalio bih se svom mentoru prof.dr.sc. Gordanu Laucu na mogućnosti obrade teme koja ima veliku komercijalnu primjenu kao i komentoru M.Sc. Genadiju Razdorovu na strpljenju i vodstvu tijekom provedbe rada

SADRŽAJ

1.UVOD

Napredak u tehnologijama prisutnim u polju biomedicinskog istraživanja rezultirao je velikom količinom podataka koju generiraju istraživanja i testiranja provedena u komercijalnim laboratorijama. Uz tako velike količine podataka, vrlo je teško kontrolirati kvalitetu procesa i generiranih rezultata. U tu je svrhu razvijen koncept Laboratory Information Management System (laboratorijski sustav upravljanja informacijama). (Melo i sur., 2010)
U suštini, LIMS je računalni program koji može upravljati laboratorijskim uzorcima, korisnicima, instrumentima te podacima. (Piggee, 2008)
LIMS naglašava osiguranje kvalitete i cilj je generirati rezultate na dosljedan i pouzdan način. On također upravlja ciklusom podataka, što uključuje prikupljanje, pohranjivanje, analizu i emitiranje izvještaja. Mada je razvijeno mnogo komercijalno dostupnih LIMS-ova, njihova visoka cijena te zatvoreni kod ograničava njihovo korištenje od strane manjih laboratorija. Također, procedure koje se vrše u različitim laboratorijima iznimno variraju, što utječe na to da je jako teško razviti LIMS koji se može podesiti potrebama svakog laboratorija. Upravo iz tog razloga, LIMS se obično razvija za potrebe samo jednog laboratorija. (Melo i sur., 2010)

Zahvaljujući podacima koji su dostupni od strane tipičnog LIMS-a, laboratorijski nadzornici kao i osoblje mogu lakša zakazati rad u laboratoriju, minimalizirati “down time”[1] te maksimalizirati veličinu šarže. Također, centraliziranost podataka omogućava lakše praćenje kvalitete. Također, lakše praćenje podataka te olakšani protok informacija daje laboratoriju mogućnost te alate za unapređenje metoda te radnih praksi što rezultira odrađivanjem većeg broja uzoraka u istoj količina vremena nego što bi to bilo moguće bez LIMS-a. (Prasad i Bodhe, 2012)

Prednosti LIMS-a su:

- Preciznost rezultata
- Efikasnost laboratorijskih zadataka
- Lakši i brži menadžment podataka

[1]"down-time" - ili "mrtvo vrijeme" predstavlja vrijeme u kojem laboratorij ne provodi analize ni obradu podataka

Glavne zadaće LIMS-a su:

- Upravljanje radnog procesa
- Čuvanje podataka
- Održavanje te upravljanje zalihama
- Izvještavanje

Tipična podjela funkcionalnosti LIMS-a: (Kolva i sur., 1994)

1. Praćenje uzorka: Omogućuje laboratorijama da prate svoje uzorke kroz različite odjele u laboratoriju pomoću računalno generiranog jedinstvenog identifikacijskog broja uzorka
2. Unos podataka: Omogućuje analitičarima da unose rezultate u LIMS te dodjele QC parametre te da obavještavaju klijente putem faxa, mail-a ili papira
3. Inventar uzoraka: Uzorci se automatski bilježe, primaju u laboratorij te se rutinski dodjeljuju laboratorijski testovi za projekte.
4. QA / QC: Omogućuje korisnicima generiranje kontrolnih grafikona i pregled analize trendova.
5. Elektronski podaci: Omogućuju automatski prijenos podataka iz analitičke instrumentacije u LIMS. Elektronski prijenos povećava produktivnost i uvelike smanjuje mogućnost pogreške u prijepisu.
6. Praćenje opreme i inventara: LIMS olakšava kalibraciju opreme, praćenje i servisiranje opreme, moguće popravke, troškove, praćenje trendova, itd.
7. Čuvanje podataka: Funkcija koja omogućuje administratoru baze podataka da upravlja bazom podataka, čuvajući praćenje popisa klijenata, zaposlenika, testova, metoda, parametara, dozvola, prioriteta itd.

1.1.Povijest LIMS-a

Do kasnih 1970-ih, upravljanje laboratorijskim uzorcima, analize i izvještavanje oduzimali su mnogo vremena te su bili skloni greškama zbog ručnih procesa. Stoga su se neke organizacije prilagodile različitom načinu prikupljanja podataka i izvještavanja, dok su neki poduzetnici započeli razvijati rješenja za komercijalno izvještavanje u obliku alata koji se temelje na informatičkim instrumentima. Osjetila se potreba za učinkovitijim sustavom i počela se razvijati sljedeća faza LIMS-a. (Gibbon, 1996)

Korisničko-definirani sustavi postali su dostupni u to vrijeme. Ovi su sustavi bili dizajnirani od strane neovisnih tvrtki za razvoj sustava sa svrhom da se koriste u posebnim laboratorijima. Neovisni LIMS-evi i komercijalni LIMS su se paralelno razvijali, a ovaj opsežni istraživački napor rezultirao je prvim komercijalnim rješenjima koja su formalno uvedena početkom 1980-ih. (Fransen i Nyrup, 1998)

LIMS-evi za obradu uzoraka i upravljanje podacima su isto toliko brojni i raznoliki kao i tvrtke koje ih koriste. U početku su LIMS-ovi razvijeni u farmaceutskoj i srodnim industrijama kao alat za osiguranje kvalitete i kontrolu kvalitete (QA[2] / QC[3]) početkom 1980-ih od strane internih razvojnih odjela ili prema ugovoru vanjskih softverskih kuća. Takvi komercijalni LIMS-i bili su vlasnički sustavi, koje su često razvijali proizvođači analitičkih instrumenata kako bi pokrenuli instrument. Ti su se LIMS-evi temeljili isključivo na miniračunalnoj ili mainframe tehnologiji, često koristeći vlasnički softver baze podataka kao svoju bazu.1982. prvi komercijalni LIMS poznat kao prva generacija (1G) LIMS predstavljen je u obliku jednog centraliziranog miniračunala, koji nudi automatizirane alate za izvještavanje. Druga generacija (2G) LIMS-ova je došla 1988. godine i koristila je relacijske baze podataka za specifična primjenjiva rješenja te je ovisila o mini-računalima. Treća generacija (3G) LIMS-eva se pojavila 1991. godine i bazirala se na otvorenim sustavima, kombinirajući jednostavno sučelje PC-a i standardizirane radne površine alata u konfiguraciji klijent /poslužitelj. Klijent /poslužitelj arhitektura podijelila je obradu podataka između niza klijenata i poslužitelja baze podataka koji pokreće cijeli ili dio sustava za upravljanje relacijskim bazama podataka (RDBMS). (http://www.umsl.edu)

LIMS je razvijen u sklopu Projekta ljudskog genoma za istraživanje genoma koji je proveo međunarodni konzorcij i smatrali su ga ključnim alatom za napredne bioinformacijske studije. Četvrta generacija (4G) LIMS-eva pojavila se 1995. godine usredotočujući se na decentralizaciju klijent[4] /server[5] arhitekture i optimiziranje dijeljenja resursa i mrežnog protoka omogućujući tako da se proces izvodi bilo gdje na mreži. Dakle, svi klijenti i poslužitelji mogu raditi na ovisno o kapacitetu pojedinačne instance. (http://www.umsl.edu)

[2]QA - "quality assurance" ili osiguranje kvalitete
[3]QC - "quailty control" ili kontrola kvalitete
[4]Klijent - označava kompjuterski program ili računalo koje šalje zahtjeve za podacima serveru
[5]Server - ili poslužitelj, označava komputerski program ili uređaj koji se zadužen za obradu zahtjeva te serviranje podataka klijentu

Web-omogućen LIMS uveden je 1996. godine, a onaj s bežičnim računalnim mogućnostima godine 1997. zbog odredbe američke FDA CFR stavke 11. koji se tiče pravila o elektroničkim zapisima i potpisima, XML se također preporučuje u industriji kao specifikacija. Prvi LIMS koji se može koristiti na mjesečnoj bazi metodom pretplate je razvijen 2002. te se temeljio na Microsoftovoj .NET platformi. (Steele i sur., 1999.)

U 2009. godini fokus je bio na više korisničkim sučeljima LIMS-a.. Paket za postavljanje i implementaciju je razvijen za LIMS koji koristi Visual Studio 2003, dok je osnovni programerski kostur činio LAMP (Linux[6], Apache[7], MySQL[8] i PHP[9]). Web sučelje koristilo je kaskadne listove oblikovanja (CSS)[10] te je dizajnirano na način da se može koristiti u web preglednicima koji ne podržavaju grafiku. Od 2010. godine te nadalje razvoj LIMS-a se usredotočio na pružanje kvalitetnijih usluga te pružajući korisno i integrirano rješenje za upravljanje podacima za učinkovito vođenje laboratorija. (Troshin i sur., 2011)

1.2.Način djelovanja LIMS-a

LIMS omogućuje znanstvenicima i tehničarima da unose detaljne bitne informacije koje se odnose na uzorke s kojima rade. Potencijalne informacije za pohranu uključuju:

- Inspekcijski broj
- Šaržni materijal
- Vrijeme i datum
- Lokaciju provedenog ispitivanja
- Dodatne podatke

Kako se uzorak probija kroz postupak testiranja, sve ove informacije čuvaju se na sigurnom i na jednom, lako dostupnom mjestu. Mnogi LIMS sustavi generirat će barkod kad se unese početni uzorak podataka tako da ga različiti članovi osoblja mogu jednostavno ponovno

[6]Linux - operacijski sustav otvorenog koda

[7]Apache - jedan od najpopularnijih web servera

[8]MySQL - relacijski sustav baze podataka otvorenog koda. SQL je skraćenica za "Structured Query Language", tj. strukturirani jezik upita

[9]PHP - ili "Hypertext Preprocessor" je programski jezik koji se koristi na serverskom dijelu aplikacija

[10]CSS - "cascading style sheets" je programski način obilježavanja teksta koji se koristi za mijenjanje izgleda dokumenata napisanih u HTML programskog jeziku. HTML ili Hyper-text Markup Language (jezik za označavanje hiperteksta) je programski jezik koji se koristi za stvaranje dokumenata i web stranica na Internetu

skenirati kako bi se dobile dodatne informacije. Svaki korisnik može skenirati isti kod kako bi osigurao da su podaci ažurni i da je postupak u tijeku. Osim što čuva podatke, LIMS može proslijediti rezultate ispitivanja drugim sustavima u lancu. LIMS znači da potreba za ručnim spremanjem podataka (što očito ostavlja prostora ljudskim greškama) više nije potrebna jer će sustav automatski spremiti podatke. To smanjuje vrijeme provedeno administrirajući laboratorij te također smanjuje mogućnost za netočnost. (Melo i sur., 2010)

1.2.1. Primjer glavne primjene LIMS-a

Kako laboratorij raste, sve se više suočava s pitanjima kako točno i brzo obraditi ogromne količine podataka za sva izvršena ispitivanja te kako se držati ispred konkurencije. Odgovor leži u korištenju najnovije tehnologije pomoću koje će se poboljšati rad laboratorija i povećati profitabilnost. LIMS-i su aplikacije za baze podataka koje se koriste za pohranu i upravljanje informacijama povezanih s laboratorijem, kao što su kupci, matrica uzorka, ispitivanja, rezultati, metode, parametri, tipovi laboratorijskog pribora, zaposlenici, lozinke, itd. U odabiru LIMS-a, postavljaju se pitanja koja uključujući i odabir određene baze podataka, jednostavnost korištenja softvera LIMS-a, kvaliteta tehničkog dobavljača, programi podrške i obuke i korištenje postojećeg hardvera ili softvera. Važno je odabrati bazu podataka otvorenog koda što omogućava jednostavan migracijski put kako se tehnologija razvija, što zaštićuje investiciju. (Kolva i sur., 1994)

Međutim, postoje određene značajke kojima svi korisnici LIMS-a trebaju biti upoznati. (Tablica 1.) Mnogi laboratoriji prelaze iz papirnatih dnevnika iz knjiga u proračunske tablice na računalu kao što je Microsoftov Excel kako bi pratili uzorke. Dok je to sjajan prvi korak, jednom kada baza podataka raste, sve je teže komunicirati s proračunskim tablicama, a relacijska baza podataka često je dobar odgovor. U odabiru LIMS-a nužno je da se LIMS podudara sa trenutnim načinom protoka uzoraka u laboratoriju, da je sustav dovoljno fleksibilan tako da se može mijenjati ovisno kako se workflow[11] mijenja te da dobavljač ima put nadogradnje dostupan kako laboratorij raste. (Kolva i sur., 1994)

[11]workflow - radni proces

Tablica 1. Pregled tipičnih značajki LIMS sustava (prilagođeno prema Kolva i sur., 1994)

Tipična funkcionalnost LIMS-a	
Tipične funkcije LIMS-a	Opis
Praćenje uzoraka	Omogućuje laboratorijima da prate svoje uzorke kroz različite odjele u laboratoriju pomoću računalno generiranog jedinstvenog identifikacijskog broja uzorka i time pruža a kompletan lanac skrbništva.
Unos podataka i generiranje izvještaja	Omogućuje analitičarima da unose rezultate u LIMS te dodjeljuju šarže koje se koriste za Quality Control. Izvještavanje se može vršiti putem elektroničke pošte ili papira. klijentima putem faksa, e-pošte ili na papiru.
Automatizacija analize uzoraka	Automatski se prijavljuje u uzorke, šalje ih u laboratorij, ispisuje naljepnice i rutinski dodjeljuje testove za projekte.
QA/QC	Omogućuje korisnicima generiranje kontrolnih grafikona i te njihov pregled. Također i pregled i analizu trendova.
Elektronički prijenos podataka	Omogućuje automatski prijenos podataka iz analitičke instrumentacije u LIMS. Povećava produktivnost i uvelike smanjuje mogućnost pogreške u transkripciji
Inventar kemikalija i reagensa	Funkcionalnost koja prati kupnju i upotrebu zaliha u laboratoriju i upravlja količinama i narudžbama, trajanjem reagensa, troškovima itd., pomažući u opskrbi i upravljanju
Upravljanje osobljem i opremom	Omogućuje korisnicima da prate zapise o obuci zaposlenika te također prati kalibraciju, popravke i troškove opreme
Održavanje	Funkcija koja omogućuje administratoru baze podataka da upravlja bazom podataka, čuvanje i praćenje popisa klijenata, zaposlenika, testova, metoda, parametara, dozvola, prioriteta, itd.

Najvažnije mogućnosti LIMS-a su:

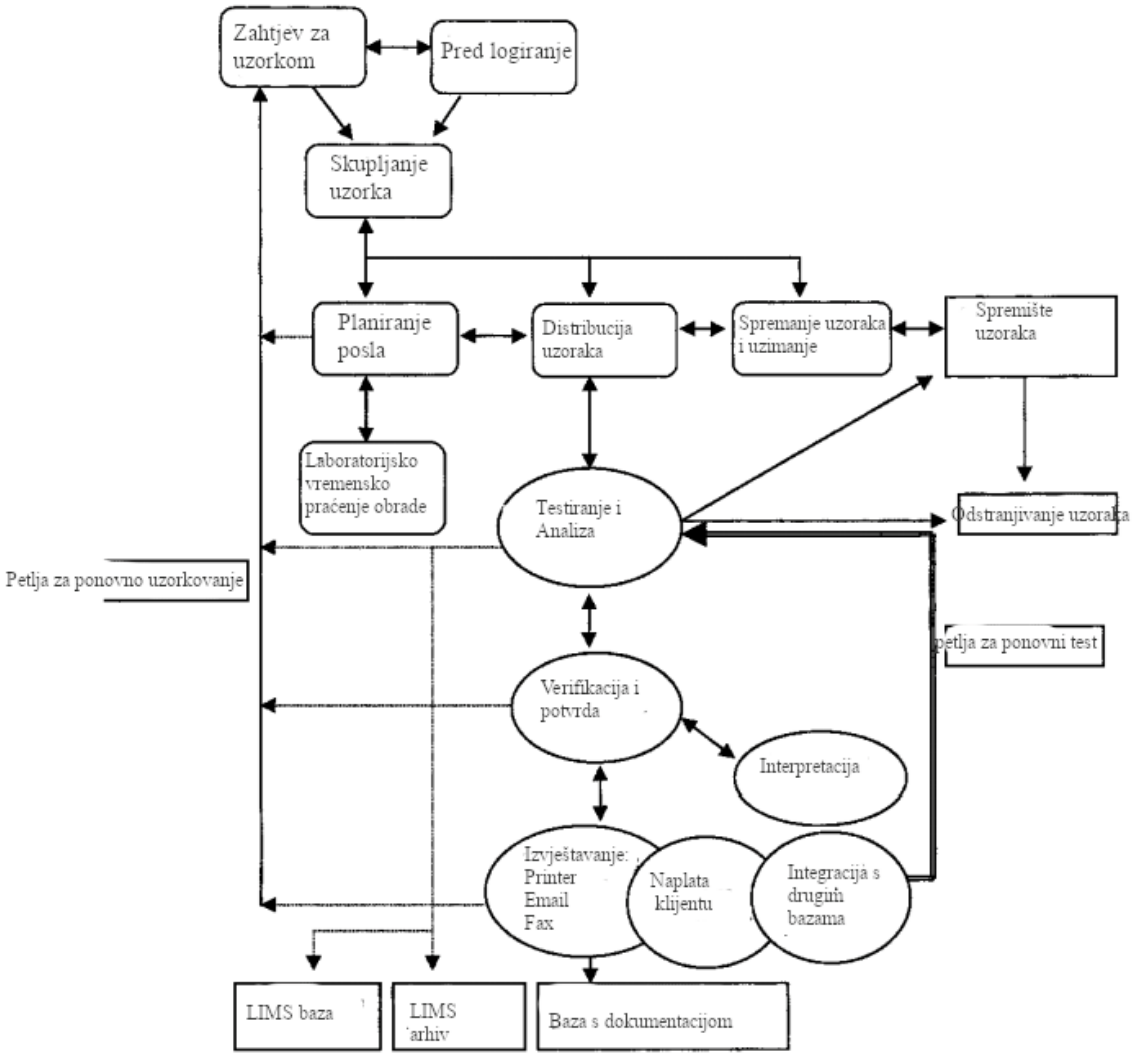
1. Upravljanje radnog procesa
2. Upravljanje uzorcima
3. Održavanje i upravljanje zalihama
4. Izvještavanje

1. Upravljanje radnog procesa

Upravljanje radnog procesa sastoji se od skupa zadataka i skupa prijelaza. Svaki pojedini zadatak sastoji se od podskupa zadataka i podskupa prijelaza. Svaki zadatak u LIMS-u označava laboratorijski proces koji će se obaviti. Sam zadatak u LIMS-u se također može odnositi na laboratorijski eksperiment. Također, nužno je definirati kontrolni tijek, tj. redoslijed izvršavanja, između početnog zadatka te odredišnog zadatka.

Objekt koji prolazi kroz dva zadatka označava dakle laboratorijski eksperiment ili proces te mora biti predstavljen prijelazom koji označava pozitivan ili negativan završetak radnje. Svaki objekt podataka prošao između dva zadatka mora biti predstavljen vlastitim (dodatnim) zadatkom.

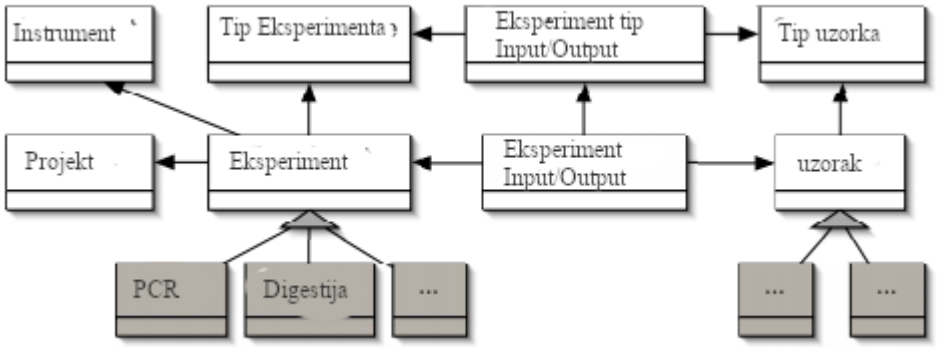
Samo ako je ishod istinit, prijelaz može biti korišten za pozitivno izvršavanje odredišnog zadatka. Upravljanje radnog procesa u LIMS-u također može odrediti određene osobe ili robote koji su zaduženi za određeni zadatak. (Gabor i Kemme, 2006)



Slika 1. Prikaz tipičnog LIMS radnog procesa (prilagođeno prema Fransen i Nyrup, 1998)

2. Upravljanje uzorcima

Ključno pitanje prilikom rada u zauzetom laboratoriju može biti poznavanje količine svih uzoraka. LIMS to rješava pružanjem sustava za praćenje predmeta, uključujući i reagense. Sustav mora biti dovoljno fleksibilan za snimanje količina laboratorijskog posuđa (npr. kutije s tanjurima) kao i sadržaj epruveta (npr. enzimi, plazmidi). Svaki uzorak ima pridruženi iznos u SI jedinicama kao što su mikrolitri. Korištenje SI jedinica omogućuje usklađivanje količina određenih uzoraka korištenih u laboratorijskom procesu. (Craig i sur.,2017)



Slika 2. Primjer definiranja radnog procesa te rada s uzorcima koristeći LIMS (prilagođeno prema Gabor i Kemme, 2006)

3. Održavanje i upravljanje zalihama

Kvalitetan LIMS omogućuje praćenje svake komponente koja sačinjava završan proizvod u laboratoriju. Tako je moguće osigurati da se količina pojedinih komponenti obračunava kroz cijeli postupak. Također, komponente se prate pomoću integriranog popisnog sustava te uključuju stavke kao što su reagensi korišteni u laboratorijskim procesima kao i sami instrumenti koje se koriste tijekom postupka stvaranja proizvoda.

Pohranjivanje predmeta na ovaj način omogućuje njihovo dijeljenje između projekata, što je ključni zahtjev kod izrade samog LIMS-a. No, konačan proizvod ne mora biti fizički predmet i može predstavljaju same podatke koje se koriste u obradi. Moguće je imati u cijelosti temeljen radni tijek koji ne proizvodi fizički predmet ni u jednom trenutku. (Craig i sur., 2017)

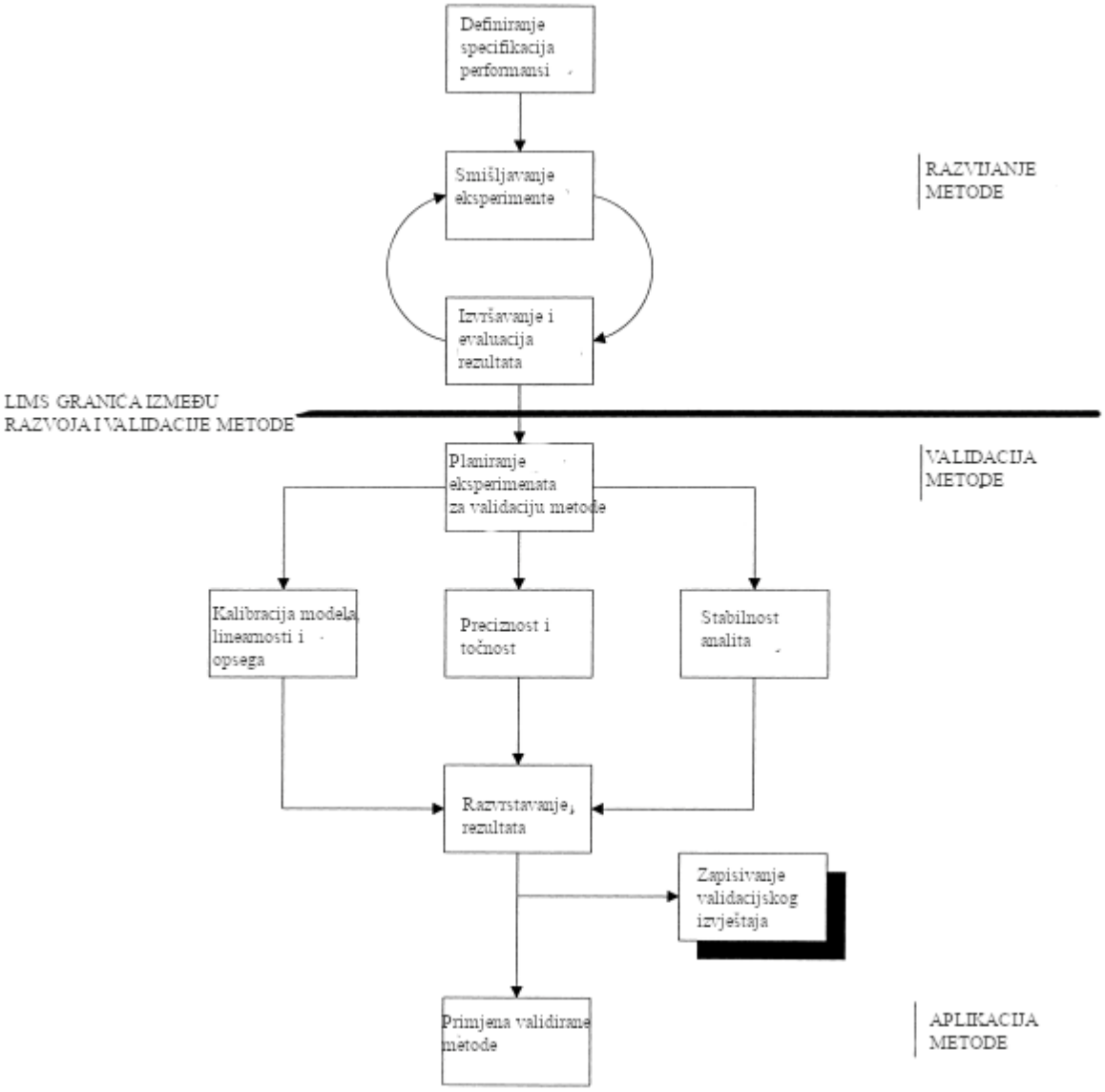
4. Izvještavanje

Uloga LIMS-ova u kvalitetnom izvještavanju je jako velika. Ne treba zaboraviti da su LIMS-evi u suštini baze podataka laboratorija koje se bave povezivanjem laboratorija s organizacijama kojoj služe.

No, LIMS-ovi su vrlo korisni i učinkoviti za upravljanje podacima dobivenim validacijom metode te izvještavanjem rezultata.

Dostupnost podataka koji se dobivaju prilikom laboratorijskog rada u LIMS-u mogu se koristiti i za (Mcdowall, 1999):

- Validaciju analitičkih metoda
- Definiranje same metode te određivanje granice gdje korištenje LIMS-a ne može pomoći lakšoj validaciji
- Lakoći upotrebe same razvijene metode te pretvaranje laboratorijskog procesa u rutinu



Slika 3. Primjer korištenja LIMS-a pri validaciji metode (prilagođeno prema McDowall, 1999)

1.3. Tehnička izvedba komunikacije servera i klijenta

Kako je vidljivo iz prethodnog poglavlja, arhitektura izrade LIMS-a u svim primjerima uključuje oblik server/klijent, s time da se različite arhitekture dijela na stupanj slojevitosti same arhitekture.

Stoga je nužno definirati komunikaciju servera sa klijentom, a industrijski standard je API[12], tj. Application Programming Interface.

[12]API - "Application Programming Interface" ili aplikacijsko programersko sučelje je sučelje preko kojeg se definira slanje i primanje podataka te time komunikacija sa klijentom i serverom

Aplikacijska programska sučelja (API) izlažu usluge ili podatke pružene softverskom aplikacijom putem skupa unaprijed definiranih resursa, kao što su metode, objekti ili URI[13]. (Stylos i sur.,2009).
Pomoću ovih resursa drugi programi mogu pristupiti podacima ili uslugama implementacije softverskih postupaka. API-i su središnji u mnogim modernim softverskim arhitekturama, jer pružaju apstrakcije visoke razine koje pogoduju programskim zadacima, podržavaju dizajn distribuiranih i modularnih softverskih aplikacija i ponovnu upotrebu koda. (Robillard i sur., 2010).

Najčešće korištena aplikacijska programska sučelja su:

- RESTful API
- SOAP
- GraphQL

1.3.1. RESTful API

RESTful je skraćenica za REpresentational State Transfer (reprezentativni prijenos stanja). Fielding je definirao reprezentativni prijenos stanja (REST) (Fielding, 2000) kao arhitektonski stil oko modela razmišljanja koji se koristi u kreiranju standarda koji opisuju Web kao što su HyperText Transfer Protocol (HTTP), Uniform Resource Identifier (URI) i HyperText Markup Language (HTML) (Severance, 2015).

Međutim, HTTP[14] ne provodi sva ograničenja definirana REST-om. Jedno ograničenje koje programer mora riješiti je hipermedija. To je jedno od četiri podograničenja i naziva se: "Hipermedia kao pokretač aplikacijskog stanja". Posebno hipermedijsko ograničenje, diskriminiraju REST od ostalih stilova. Razumijevanje i provođenje ograničenja hipermedije je stoga kritični dio u procesu izgradnje REST API-ja. (https://roy.gbiv.com/)

[13]URL - "Uniform Resource Identifier" ili uniformirani identifikator resursa znana kao i web adresa na kojoj je specificirana lokacija resursa na web serveru

[14]HTTP - "Hypertext Transfer Protocol" ili protokol transfera hiperteksta je temeljni protol World Wide Web-a kojim se definira kako se hipertekst formatira te šalje između klijenta i servera

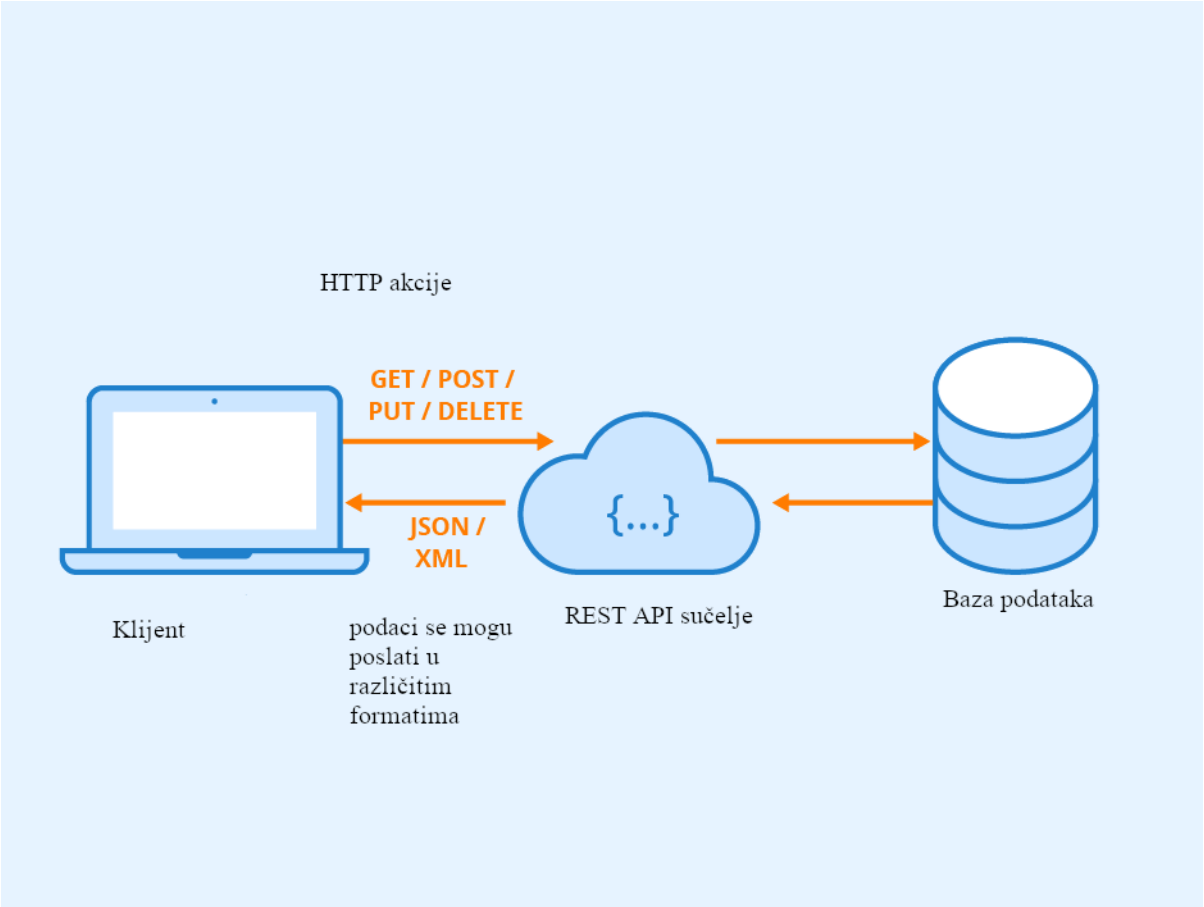
Središnji koncept koji se odnosi na dizajn sučelja u REST-u je resurs (Fielding, 2000). Resurs je apstrakcija oko određenog podatka. Da bismo razumjeli koncepte operacija u REST-u je važno razlikovati koncept resursa i njihovu zastupljenost i radnje. Resurs je koncept koji može imati bilo koji broj predstavnika u određenoj točki u vremenu, što znači da reprezentacija resursa može varirati. Ova varijacija ne može doći samo u obliku različitog stila kodiranja, poput upotrebe proširivog jezika za označavanje (XML[15]) umjesto JSON-a (JavaScript Object Notation), već zapravo različite informacije. Ono što se ne smije promijeniti je semantika. Na primjer, resurs koji opisuje najbolje osobe može se odnositi na različite osobe ovisno o vremenu. Informacija koja predstavlja resurs može biti promijenjena, ali ipak predstavlja popis osoba, ostavljajući semantiku resursa netaknutu. Uz to, dostupne radnje mogu varirati. (Fielding, 2000)

Vodeća načela REST metodologije: (Massé, 2012):

- Klijent-poslužitelj - Odvajanje briga o korisničkom sučelju od brige o pohrani podataka poboljšava se prenosivost korisničkog sučelja na više platformi te se omogućava skalabilnost
- Bez stanja - svaki zahtjev od klijenta do poslužitelja mora sadržavati sve informacije potrebne za razumijevanje zahtjeva i ne može iskoristiti bilo koji pohranjeni kontekst na poslužitelju. Stanje sesije se stoga u potpunosti zadržava na klijentu.
- Predmemoriranje - Ograničenja predmemoriranja zahtijevaju da podaci u odgovoru na zahtjev budu implicitno ili eksplicitno označeni kao predmemorirani ili ne-predmemorirani. Ako je odgovor predmemoriran, tada će klijentska predmemorija dobiti pravo da ponovo koristi te podatke odgovora za kasnije, jednake zahtjeve.
- Uniformno sučelje - Primjenom programskog inženjerskog načela općenitosti pojednostavljuje se cjelokupna arhitektura sustava i poboljšava se vidljivost interakcija. REST je definiran s četiri ograničenja sučelja: identifikacija resursa; manipuliranje resursima putem reprezentacija; poruke s samim opisom i hipermedija kao motor stanja primjene.

[15]XML - "eXtensible markup Language" ili proširivi jezik za označavanje predstavlja jedan od oblika slanja podataka putem Interneta

- Slojeviti sustav - Slojeviti sistemski stil omogućava arhitekturi da se sastoji od hijerarhijskih slojeva ograničavanjem ponašanja komponenata tako da svaka komponenta ne može "vidjeti" izvan neposrednog sloja s kojim komuniciraju.
- Kod na zahtjev (neobavezno) - REST omogućuje proširivanje funkcionalnosti klijenta preuzimanjem i izvršavanjem koda u obliku skripti. To pojednostavljuje klijente smanjenjem broja značajki koje je potrebno prethodno implementirati.



Slika 4. Pregled RESTful APi sučelja - klijent šalje zahtjeve putem predefiniranih HTTP akcija, REST API sučelje ih obrađuje u komunikaciji s bazom podataka te šalje klijentu u JSON ili XML formatu

1.3.2. SOAP

SOAP (skraćenica za Simple Object Access Protocol) je specifikacija protokola za razmjenu poruka koji se koristi u razmjeni strukturiranih informacija u implementaciji web usluga koje koriste računalne mreže.
SOAP je stari protokol koji se oslanja na XML za propusne web usluge. Za komunikaciju, SOAP preporučuje WDSL[16] (Web Services Description Language), koji je razvijen od World Wide Web konzorcija. Jezik opisa (WSDL) djeluje kao sučelje između web aplikacije i sadrži podatke o web uslugama poput naziv metoda, parametara te kako im pristupiti. To je dio UDDI[17] (Universal Description, Discovery and Integration), koji je okvir za definiranje i deklariranje web usluge. Također, bitno je napomenuti da web usluga koja koristi SOAP omogućuje otkrivanje funkcionalnost koda preko mreže. (Soni i Ranga, 2019)

1.3.3. GraphQL

GraphQL (Graph Query Language[18]) je tehnologija koja je u kreirana u Facebooku 2012. godine. Nacrt specifikacije otvoren je 2015. godine (https://github.com). Opisan je kao "jezik upita za vaš API" i bio je izgrađen kao odgovor na problem s performansama tijekom Facebook-ovog prelaska na izvorne mobilne aplikacije (native mobile applications) (https://github.com).
U osnovi, to je tehnologija za izgradnju API-ja i zamišljena je kao alternativa REST-u. Bez obzira na ime, ona ne omogućuje samo preuzimanje podataka s poslužitelja, već i njihovo modificiranje. Sljedeći odjeljak objašnjava kako GraphQL radi.

Schema
Shema je od centralne važnosti za GraphQL. Opisuje dostupne vrste i njihove odnose, kao i ulazne točke za klijente; upite i mutacije. Uskličnik označava ne-nulabilnost, tj. polje ne može biti ništavno. Uglate zagrade oko definicije tipa označavaju niz zatvorenog tipa. Parametri kao u upitima i mutacijama su imenovani i mogu se stoga specificirati bilo kojim redoslijedom

Upiti
Da bi se dobili podaci iz usluge GraphQL, njemu se šalje upit. Upiti mogu biti u obliku fragmenata, čime se može dobiti rekurzivni oblik upita, tj. koristi se ponovno jedno polje

[16]WDSL - Web Services Description Language ili opisni jezik web servisa
[17]UDDI - Universal Description, Discovery and Integration, tj. Univerzalna opisivost, otkrivanje i integracija
[18]Graph Query Language - Grafovski jezik upita

upita da bi se dobiveni upit proširio. Korištenje fragmenata je slično “spread”[19] operatoru u ECMA-a skripti 2015. (http://www.ecma-international.org/)
Prilikom slanja upita, GraphQL zahtijeva da su sva tražena polja primitivne vrste kako bi se poboljšala predvidljivost rezultata. (https://github.com)

Mutacije

Mutacije su slične upitima, samo što imaju mogućnost da uzrokuju nuspojave, no to nije definirano GraphQL specifikacijom, već je samo uobičajenost (https://github.com). Mutacije se stoga koriste za promjenu podataka na poslužitelju. Mutacije vraćaju vrijednosti, baš kao i upiti. Stoga klijent može zatražiti podatke na temelju povratne vrijednosti mutacije.

[19]spread operator - ili operator proširenja označava se u ECMA skripti kao ... i označava prebacivanje liste podataka u niz i obrnuto

Tablica 2. Primjer definiranja scheme s upitima i mutacijama. Korišten je programski jezik JavaScript. U ovom primjeru scheme stvoreni su tipovi Korisnik, Eksperiment kao i input tipovi koji omogućuju definiranje upita i mutacija.

```
const {
    buildSchema
} = require('graphql')

module.exports = buildSchema(`

    type Eksperiment {
        _id: ID!
        titula: String!
        opis: String!
        datum: String!,
        kreator: User!
    }

    type Korisnik {
        _id: ID!
        email: String!
        lozinka: String,
        kreiraniEksperimenti: [Eksperiment!!]
    }

    input EksperimentUnos {
        naziv: String!
        opis: String!
        cijena: Float!
        datum: String!
    }

    input KorisnikInput{
        email: String!
        lozinka: String!
    }

    type RootQuery {
        eksperiment: [Eksperiment!!]
    }

    type RootMutation {
        kreirajEksperiment(eksperimentUnos: EksperimentUnos): Eksperiment
        kreirajKorisnika(korisnikUnos: KorisnikUnos): Korisnik
    }
```

1.4. Dizajn aplikacije

Što se tiče dizajna aplikacije on najčešće spada u iduće tri kategorije (Ming i Tang, 2010):
1.MVC - Model View Controller (Model Prikaz Upravljač)
2. MVP - Model View Presenter (Model Prikaz Izlagač)
3. MVVM - Model View View-Model (Model Prikaz Prikaz-Model)

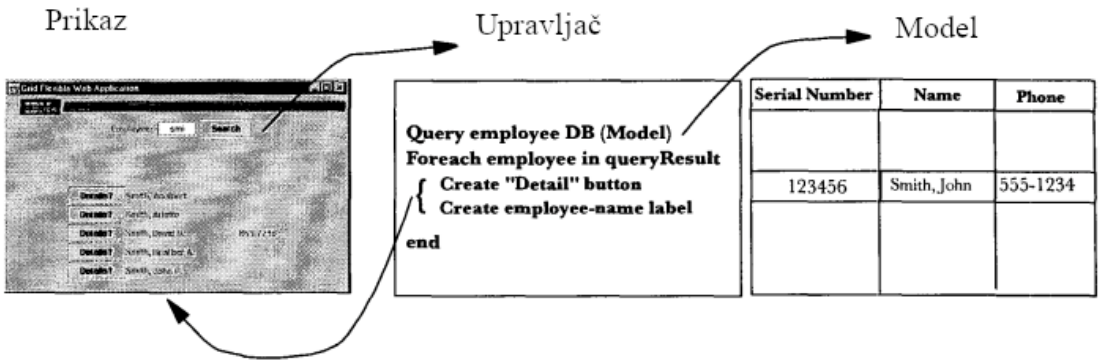
1.4.1. MVC - Model View Controller

Široko poznati obrazac dizajna MVC, tj Model/View/Controller (Model/Prikaz/Upravljač) koristan je kao način za arhitekturu interaktivnih softverskih sustava. Također poznat kao PAC, tj Prezentation/Abstraction/Control (Prezentacija/Abstrakcija/Kontrola).(Sarcar i Shimanthoor, 2018)

Model
Model je dio aplikacije koji sadrži informacije predstavljene prikazom te logiku koja mijenja informacije sukladno korisnikovim akcijama. (Leff i Rayfield, 2001)
Model upravlja ponašanjem i podacima korisnika aplikacijske domene te odgovara na zahtjeve za informacijama o njenom stanju (koji se prikazuju putem Prikaza) i odgovara na upute za promjenu stanja (kojim upravlja Upravljač).

Prikaz
U MVC-u, Prikaz prikazuje informacije za korisnike i zajedno s Upravljačem koji obrađuje korisnikove interakcija te obuhvaća korisničko sučelje aplikacije. (Leff i Rayfield, 2001)

Upravljač
Upravljač interpretira unose miša i tipkovnice od korisnika, upravljajući modelom i / ili prikazom koji će se promijeniti po potrebi.



Slika 5. Primjer podjele na Prikaz, Upravljač te Model kao i pregled toka informacija. (prilagođeno prema Leff i Rayfield, 2001)

1.4.2. MVP - Model View Presenter

MVP (Model/View/Presenter, tj. Model/Prikaz/Izlagač) uzorak sličan je MVC obrascu. Izvedeno je iz MVC uzorka, pri čemu je Upravljač zamijenjen Izlagačem. Model-View-Presenter (MVP) je dizajnerski obrazac razvijen za prezentacijski sloj softverskih aplikacija (Zhand i Luo, 2010). Obrazac je izvorno razvijen u Taligentu 1990-ih i najprije je implementiran u C ++ i Javi.

Ovaj obrazac dijeli aplikaciju na tri glavna aspekta: model, prikaz i izlagač: (Zhand i Luo, 2010)

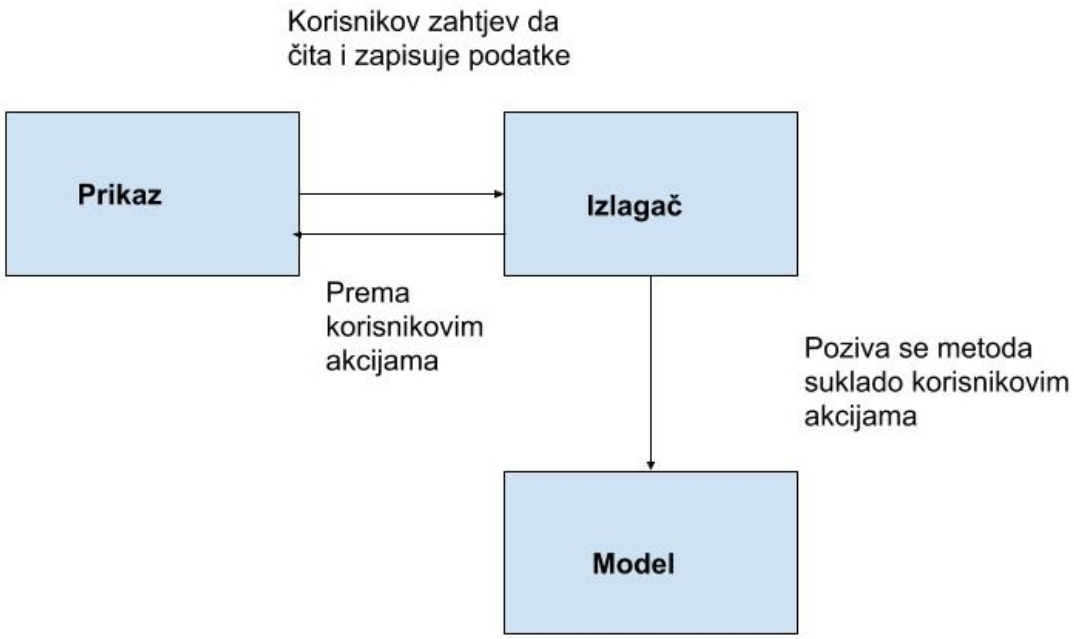
Model
Model je komponenta koja čuva podatke, stanje i poslovnu logiku; model samo izlaže skupinu sučeljnih usluga za Izlagač i skriva interne detalje.

Prikaz
Prikaz je korisničko sučelje, ono prima radnju korisnika te komunicira sa Izlagačem za postizanje potreba korisnika, a prikazuje relevantne informacije korisnicima.

Izlagač

Izlagač se nalazi između Prikaza i Modela, on prima informacije od Prikaza te zatim šalje naredbe do Modela. Zatim dobivene rezultate prikazuje putem Izlagača kroz definirano sučelje.

U MVP dizajnu dizajna, Izlagač manipulira modelom i također ažurira pogled. U MVP-u Prikaz i Izlagač potpuno se razdvajaju i međusobno komuniciraju putem sučelja. Jer iako je jednostavnije odvajanje modulacije pogleda, jednostavnije je testiranje aplikacija. (Zhand i Luo, 2010)



Slika 6. Reprezentacija Model-Prikaz-Izlagač te primjer toka podataka

1.4.3. MVVM - Model View View-Model

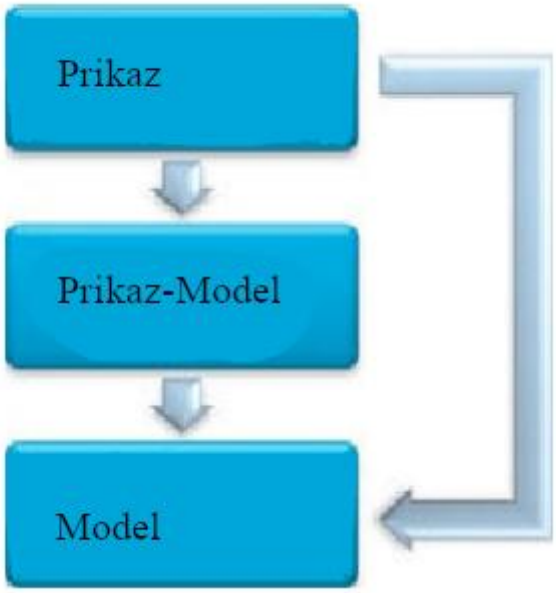
U središtu dizajna MVVM (Model/Prikaz/Prikaz-Model) nalazi se odvajanje logike Prikaza od njegovog izgleda. Umjesto velike količine koda odgovorne za Prikaz, velik dio ovog koda izdvojit će se u klasu Prikaz-Model koji je odvojen od samog Prikaza. Ta klasa Prikaz-Model javno izlaže podatke i operacije jednog ili više vlasničkih pregleda. Rezultat toga je da Prikaz mora imati vrlo malo koda. Zatim se koristi vezivanje podataka i naredbi/ponašanja kao ljepilo za povezivanje Prikaza i Prikaza-Modela zajedno.

Obrazac dizajna MVVM sastoji se od tri jezgra:

- Model
- Prikaz
- Prikaz-Model

Na prvi pogled čini se sličan tradicionalnom troslojnom obrascu, koji se sastoji od sloja Prikaza, sloja poslovne logike i podatkovnog sloja. Iako su prilično različiti obrasci i služe različitim potrebama, možemo usporediti slojeve u MVVM obrascu dizajna s slojevima troslojnog uzorka u sljedećim aspektima: (Anderson, 2012)

- Sloj Prikaza odgovara sloju prezentacije.
- Sloj Modela odgovara sloju podataka.
- Prikaz-Model sloj odgovara sloju poslovne logike



Slika 7. Prikaz MVVM modela te pregled toka podataka (prilagođeno prema Anderson, 2012)

1.5. Pregled mogućih baza podataka

Tradicionalni sustavi za pohranu baza podataka temeljeni su na relacijskom modelu. Oni su nadaleko poznati kao SQL baze podataka nazvane prema jeziku kojim se ostvaruju upiti. (Kline i sur., 2009).

Posljednjih nekoliko godina, međutim, postoje ne-relacijske baze podataka te im je dramatično porasla popularnost. Te su baze podataka obično poznate kao NoSQL baze podataka, te se time jasno razlikuju od tradicionalnih SQL baza podataka. Većina ovih baza temelje se na pohranjivanju jednostavnih parova ključ-vrijednost,a ta jednostavnost dovodi do brzine. (Li i Manoharan, 2013)

1.5.1. Relacijske baze podataka

U 1970-ima, IBM je razvio proizvod nazvan SEQUEL ili Structured English Query Language, koji je na kraju postao SQL (Structured Query Language), tj. strukturni jezik upita.
IBM je zajedno s drugim dobavljačima relacijskih baza podataka želio standardiziranu metodu za pristup i manipuliranje podacima u relacijskoj bazi podataka. Tijekom desetljeća, mnogi su konkurentni jezici omogućili programerima pristup i manipulaciju podacima. Međutim, samo ih se nekoliko lako usvaja te je postalo tako široko prihvaćeno kao SQL. Programeri sada imaju korist od učenja jezika koji je uz manja prilagođavanja primjenjiv na široku paletu baza podataka, aplikacija i proizvoda.
Četiri najpopularnije implementacije trenutnog SQL standarda, SQL99 (poznat i kao SQL3) su (Kline i sur., 2009):

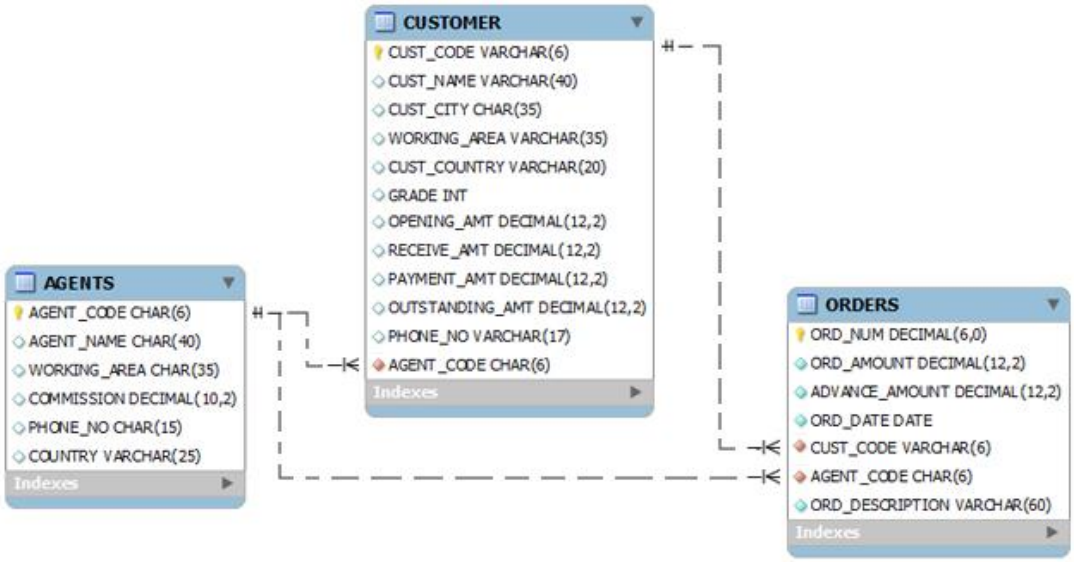
- Microsoftov SQL Server,
- MySQL,
- Oracle i
- PostgreSQL.

RDBMS[20] (Relational Database Management System) je definiran kao sustav čiji korisnici prikazuju podatke kao zbirku tablica međusobno povezane zajedničkim vrijednostima podataka. Podaci se pohranjuju u tablice, a tablice su sastavljene od redaka i stupaca. Tablice neovisnih podataka mogu biti povezani (ili povezani) jedan s drugim ako svaki ima skupove podataka (zvani ključevi) koji predstavljaju istu vrijednost podataka. (Kline i sur., 2009)

[20]RDBMS - sustav upravljanja relacijskim bazama podataka

Tablica 3. Primjer SQL komande za kreiranje tablice, svako polje ima definirani oblik. Int označava cjelobrojnu vrijednost, a varchar označava tekstualno polje definirane duljine

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```



Slika 8. Prikaz povezivanja tablica putem stranih i primarnih ključeva. Ikona žutog ključa označava primarni kjuč tablice. Nužno je da je to polje jedinstveno u tablici. Crveni kvadrat označava strani ključ, koji je u suštini primarni ključ druge tablice te se koristi za međusobno povezivanje tablica prilikom strukturiranja upita. (prilagođeno prema https://www.w3resource.com/, pristupljeno 23.3.2020)

1.5.2 NoSQL baze podataka

Sa povećanjem pristupa Internetu i dostupnošću jeftinog prostora za pohranu, ogromne količine strukturiranih, polustrukturiranih i nestrukturiranih podataka prikupljaju se i pohranjuju u razne aplikacije. Takvi se podaci obično navode kao Big data (Veliki podaci). (Warden, 2011)

Obrada tako ogromne količine podataka zahtijeva brzinu, fleksibilne sheme i distribuirane (tj. necentralizirane) baze podataka. Upravo za tu svrhu su nastale NoSQL baze podatak (Not only SQL). NoSQL baze podataka postale su preferirane za rad s velikim podacima za koje tvrde da udovoljavaju zahtjevima korisnika. To također dovodi do porasta broja NoSQL baza podataka. (Han i sur., 2011)

Glavne prednosti NoSQL-a su sljedeći aspekti:

- Brzo čitanje i pisanje podataka
- Potpora masovnoj pohrani
- Laka proširivosti
- Niski troškovi

U međuvremenu, NoSQL ima nekih nedostataka, jer ne podržava SQL, što je industrijski standard. Također nedostatnost osiguravanja transakcija, izvještaja i drugih dodatnih značajki, te se generalno smatraju nisu dovoljno zreli za komercijalnu upotrebu. (Han i sur., 2011)

1.5.2.1. Vrste baza podataka NoSQL

1. Baze podataka dokumenta

Spajaju svaki ključ sa složenom strukturom podataka koja se naziva i dokument. Dokumenti mogu sadržavati mnogo različitih parova ključ-vrijednost ili parova nizova ključeva ili čak ugniježđenih dokumenata. Primjer je MongoDB. (Bradshaw i sur., 2019)

2. Pohrana grafova

Koriste se za pohranu podataka o mrežama podataka, poput društvenih veza. Grafovske baze uključuju Neo4J i Giraph. (Li i Manoharan, 2013)

3. Pohrana ključeva i podataka:

Najjednostavnije NoSQL baze podataka. Svaka pojedina stavka u bazi podataka pohranjuje se kao naziv atributa (ili 'ključ'), zajedno s njegovom vrijednošću. Primjeri ključ-podatak baza su Riak i Berkeley DB. Neke baze poput Redisa, omogućuju da svaka vrijednost ima vrstu, što dodaje funkcionalnosti. (Han i sur., 2011)

4. Široko stupčane baze

Poput Cassandra i HBase optimizirane su za upite u velikim skupima podataka i pohranjuju stupce podataka zajedno, umjesto redaka (Győrödi i sur., 2015)

Tablica 4. Primjer definiranja korisnika u MongoDB-u koristeći Mongoose kao ODM[21] knjižnicu. Programski jezik kojim se upravlja bazom podataka je JavaScript. Svako definirano polje ima definirani oblik. Podaci se spremaju u jednostavnom obliku ključ-vrijednost. Ovim primjerom je definirana schema korisnika programa, a navedeni podaci se spremaju u jedan dokument koji je povezan sa korisnikom putem jedinstvenog identifikatora koji MongoDB automatski generira

[21]ODM - Object Data Modelling ili objektno modeliranje podataka. Označava programske knjižnice koje se koriste za lakše upravljanje bazama podataka povečavajući razinu apstrakcije

```
const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: [true, 'Molimo navedite svoje ime']
    },
    email: {
        type: String,
        required: [true, 'Navedite ispravnu email adresu'],
        unique: true,
        lowercase: true, // prebacivanje email-a u lowercase za lakšu validaciju
        validate: [validator.isEmail, 'Molimo navedite ispravnu email adresul']
    },
    photo: {
        type: String,
        default: 'default.jpg'

    },
    role: {
        type: String,
        enum: ['korisnik', 'nadzornik', 'voditelj projekta', 'administrator'],
        default: 'korisnik',
    },
    password: {
        type: String,
        required: [true, 'Molimo unesite svoju lozinku],
        minlength: 8,
        select: false // ovom postavkom osiguravamo da se lozinka ne salje klijentu
    },
    passwordConfirm: {
        type: String,
        required: [true, 'Molimo potvrdite svoju lozinku'],
        validate: {
            validator: function (el) {
                return el === this.password;
                // vraća true ili false ovisno o ispravnosti lozinke

            }
        },
        message: 'Lozinke se ne podudaraju'
    },
    passwordChangedAt: Date,
    passwordResetToken: String,
    passwordResetExpires: Date,
    active: {
        type: Boolean,
        default: true,
        select: false
    }
});
```

2. OBRAZLOŽENJE TEME

Laboratory Information Management Systemi su nezaobilazni dio radnog procesa svake dovoljno razvijene biotehnološke tvrtke. Sama količina podataka kojima se upravlja zahtijeva pronalaženje optimalnog računalnog rješenja za njihovu obradu i korištenje. Uzevši u obzir da većina troškova laboratorija odlazi na troškove osoblja, dolazi se do zaključka da je vrijeme zaposlenika najveći ukupni trošak same tvrtke stoga je potrebno pronaći najoptimalniji način njihovog korištenja.
Upravo iz tog razloga su razvijeni LIMS-a, da automatiziraju te standardiziraju obrasce provođenje rutinskih zadataka te da smanje mogućnost pogreške.

Prednosti LIMS-a su:

- Preciznost rezultata
- Efikasnost laboratorijskih zadataka
- Lakši i brži menadžment podataka

Glavne zadaće LIMS-a su:

- Upravljanje radnog procesa
- Čuvanje podataka
- Održavanje te upravljanje zalihama
- Izvještavanje

Mada postoje komercijalna rješenja, biotehnološke mlade start-up tvrtke su jedinstvene te je često potrebno stvoriti podesiva rješenja koje najviše pogoduju već organski stvorenom radnom procesu tvrtke.
Upravo iz tog razloga ovaj diplomski rad služi kao pregled mogućeg tehničkog razvoja upravo takve aplikacije te uzima u obzir industrijske standarde u razvoju sličnih aplikacija.

Svrha rada je razdvojiti na sastavne dijelove razvoj LIMS-a koristeći se dostupnom literaturom te primjerima sličnih idejnih rješenja.
Sam zadatak podijeljen je u nekoliko etapa:

- Odabir programskog jezika te već stvorenih framework-a
- Odabir arhitektonske izvedbe LIMS-a te dizajn aplikacije
- Definiranje načina komunikacije između serverskog dijela aplikacije te klijenta

- Odabir baze podataka
- Odabir klijentskog programskog jezika te odabir frameworka-a

Kako biotehnološke tvrtke upravljaju sa jako velikom količinom informacija nužno je odabrati najbolju bazu podataka koja bi služila kao centar pohrane podataka s kojom komunicira serverski dio aplikacije.
Također analizirani su i mogući programski okviri u najpopularnijim jezicima koji bi omogućili najbrži, najefikasniji i najekonomičniji razvoj same aplikacije.
Također, cilj ovog diplomskog rada je i ukazati na porast efikasnosti koji se može postići korištenjem računalnog rješenja za vođenje rutinskih zadataka biotehnološke tvrtke te može služiti kao predložak izrade samostalnog rješenja LIMS-a za manje biotehnološke tvrtke.

3. MATERIJALI I METODE

U ovom radu je cilj bio prikazati prednosti korištenja Laboratory Information Management Systema (LIMS-a) kod biotehnoloških laboratorija kao i analizirati samu izvedbu LIMS-a podijelivši je na manje podzadatke te pronaći najbolje izvedbeno rješenje za svaki podzadatak.

Prilikom odabira najboljeg idejnog rješenja za svaki pojedini dio LIMS-a uzimali su se u obzir parametri iznimno bitni za biotehnološke tvrtke kao što su:

- Sigurnost podataka
- Cijena izrade
- Pouzdanost rješenja
- Mogućnost povezivanja sustava s drugim laboratorijima
- Jednostavnost izvedbe

Glavni materijali korišteni prilikom analize teme ovog diplomskog rada su već objavljene knjige, znanstveni radovi te ostala znanstvena glasila koja su detaljno analizirala svaki pojedini aspekt izrade LIMS-a te se na temelju tih dostupnih alata moglo doći do informiranog zaključka o najboljoj izvedbi.

Metoda analize problema je stoga primarno deduktivna, gdje se na temelju već dostupnih velikih količina informacija dolazi do zaključka.

Problematika nalaženja najboljeg idejnog rješenja te tehničke izvedbe je podijeljen u dijelove te je svaki zasebni dio LIMS-a analiziran zajedno s prednostima i manama te je na kraju prezentirano rješenje.

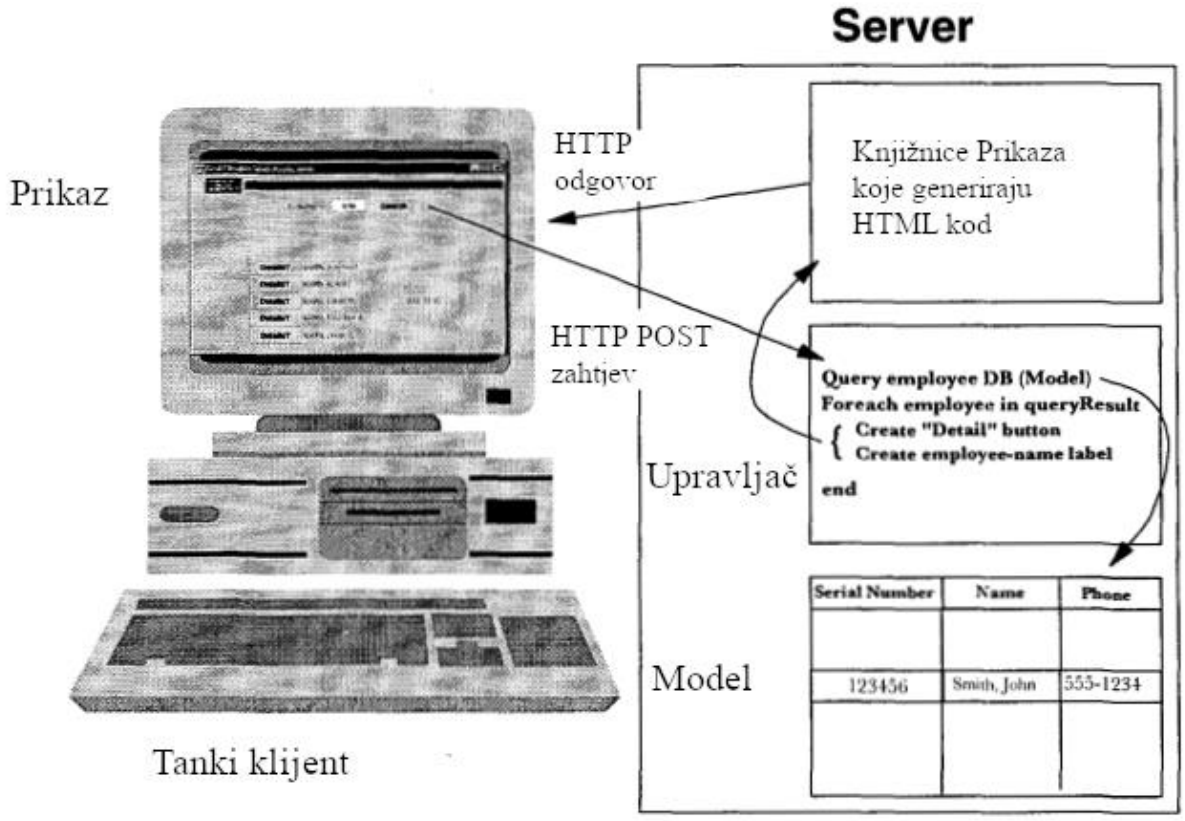
4. REZULTATI I RASPRAVA

Ovo poglavlje je podijeljeno u 5 podjedinica, a svrha svakog potpoglavlja je dati idejno rješenje za jedan dio tehničke izvedbe LIMS-a. Konačan zaključak je na kraju poglavlja prezentiran te su zaključci objašnjeni.

4.1. Odabir arhitekture izvedbe LIMS-a

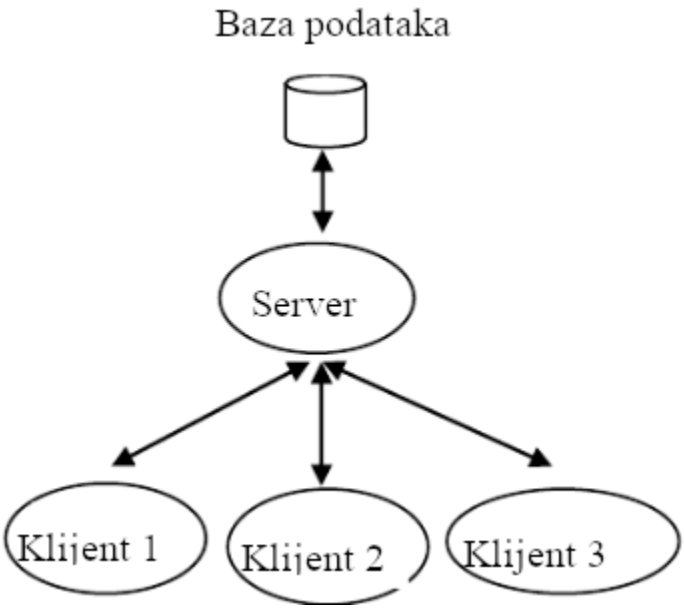
Kako je i daleko opsežnije navedeno u uvodu, tehnička izvedba LIMS-a se tijekom godina mijenjala te je uključivala različite arhitekture i modele distribucije.
Najčešće arhitekture LIMS-a su:

- Debeli klijent
- Tanki klijent
- Web-omogućen
- LIMS na Oblaku te SaaS LIMS



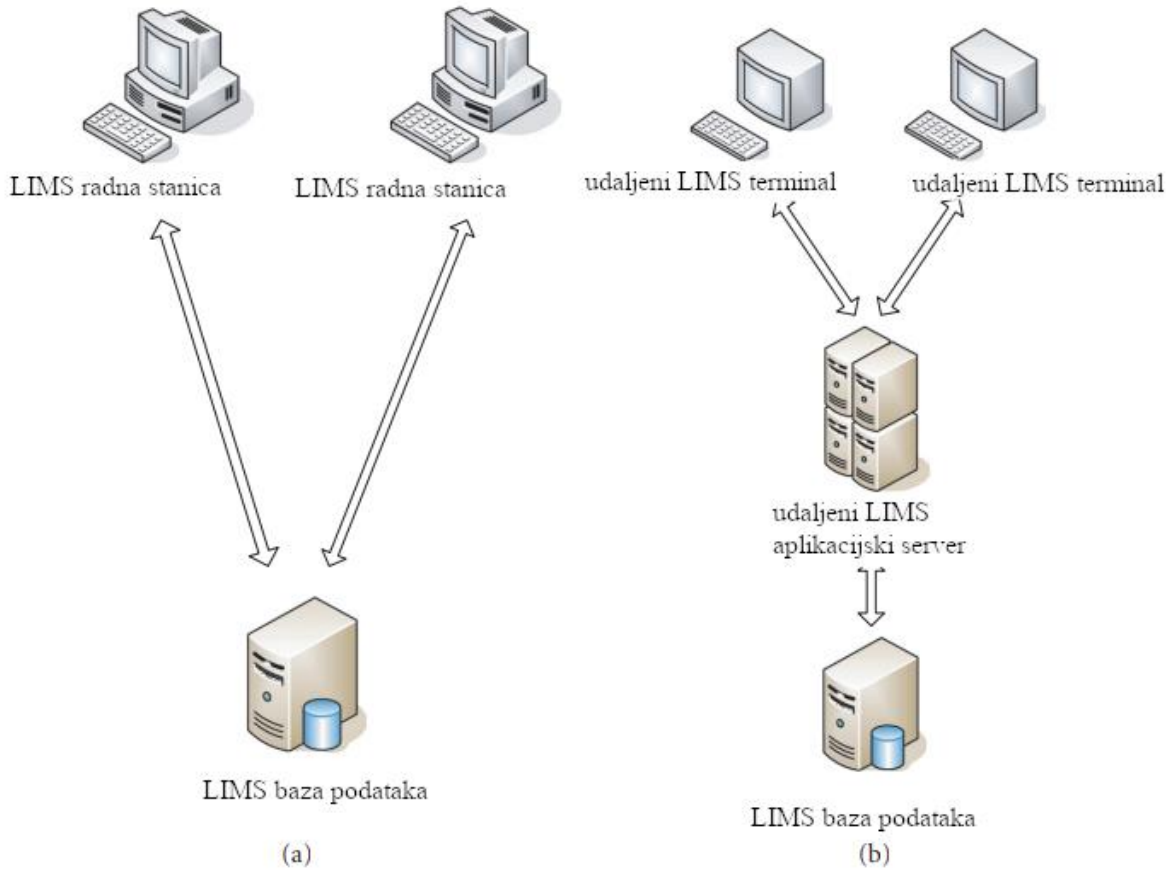
Slika 9. Primjer tankog klijenta koji koristi MVC (Model Prikaz Upravljač) dizajn aplikacije. Na klijentu se nalazi samo Prikaz, dok je poslovna logika na udaljenom serveru. (prilagođeno prema Leff i Rayfield, 2001)

Svaki od navedene 4 arhitekture podrazumijeva nekakav oblik izvedbe klijent/poslužitelj.



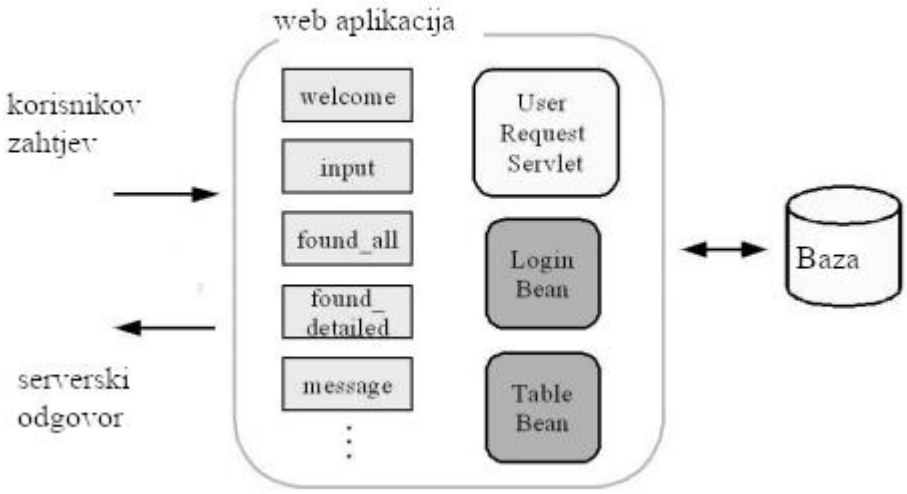
Slika 10. Primjer međuprocesne komunikacije između servera te klijenta (prilagođeno prema Oluwatosin, 2014)

Najveća razlika između debelog i tankog klijenta je ta što se poslovna logika u debelom klijentu nalazi na računalu korisnika, dok tanki klijent poslovnu logiku zadržava na poslužitelju.



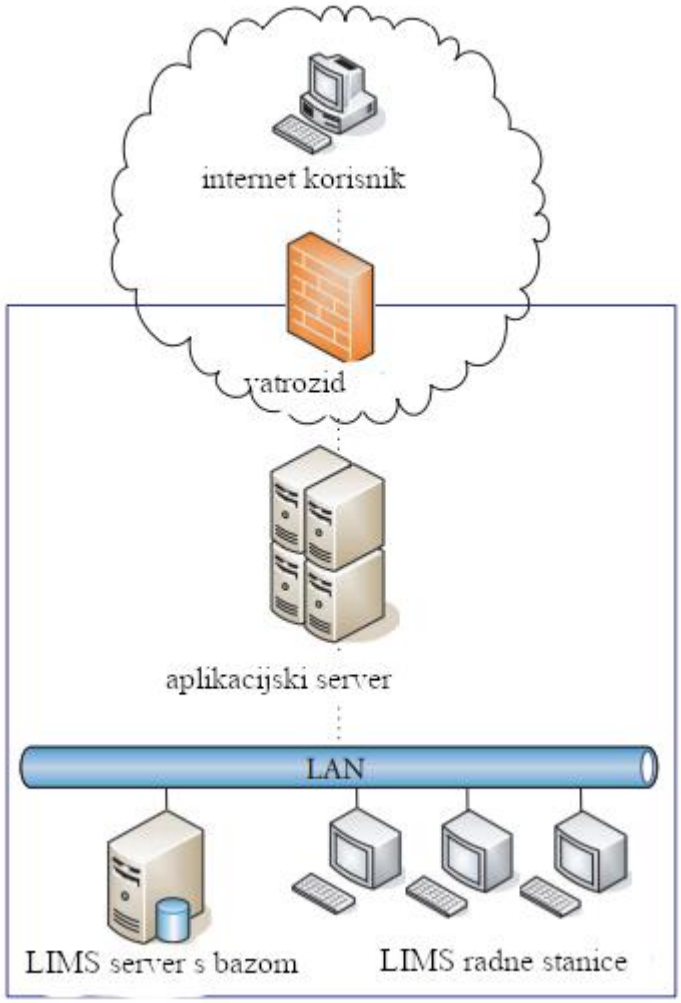
Slika 11. Usporedba između debelog a) i tankog klijenta b). Razlika je u tome što u debelom klijentu svaka radna stanica posjeduje cijelu LIMS aplikaciju. To može uzrokovati probleme sa sinkronizacijom podataka između klijenata. Kod tankog klijenta, klijent šalje i prima podatke, dok se oni obrađuju na udaljenom LIMS aplikacijskom serveru (prilagođeno prema Ulma i Schlabach, 2005)

Obzirom da laboratoriji sve više surađuju, najbolje arhitektonsko rješenje uključivalo bi mogućnost komunikacije putem web-a.



Slika 12. Izgled web aplikacije koja obrađuje korisnikove zahtjeve te šalje odgovor s time da komunicira sa bazom podataka. Web aplikacija omogućava istovremeno pristupanje podacima kao i udaljenu komunikaciju putem Interneta (prilagođeno prema Gabor i Kemme, 2006)

Tehničke razlike između web-omogućenog LIMS-a te SaaS LIMS-a su zanemarive. Može se smatrati da je najveća razlika što su Saas LIMS-ovi najčešće komercijalne prirode te se iznajmljuju laboratorijima uz redovnu naknadu.
Web-omogućen LIMS najčešće označava LIMS koji se nalazi u vlasništvu laboratorija. U slučaju potrebe može se dostupnost serveru osigurati putem vatrozida.



Slika 13. U ovom primjer web-omogućenog LIMS-a vatrozid onemogućava korisnicima Interneta da slobodno pristupaju LIMS aplikacijskom serveru. Unutar LAN[22]-a (Local Area Network) pristup je omogućen svim korisnicima. Na taj način se osigurava pristup web-omogućenom LIMS-u jedino unutar lokalne mreže (prilagođeno prema Ulma i Schlabach, 2005)

Sama arhitektura web-omogućenog i SaaS LIMS-a je u suštini identična. Označava podjelu na server, bazu podataka te klijentska računala dok se komunikacija između pojedinih dijelova LIMS-a može odvijati pomoću middleware[23]-ova.
Stoga, predloženo arhitektonsko rješenje LIMS-a bi bilo web-omogućeno, čime laboratorij osigurava veću kontrolu nad svojim podacima, također time se omogućava suradnja između udaljenih laboratorija te postiže centralna kontrola poslovne logike.

[22]LAN - Lokalna mreža je mreža povezanih računala koji se nalaze na istoj lokaciji. Povezani su putem ethernet kabela

[23]middleware - ili posrednički softver, označava posrednički kod između dva računalna procesa

Također, jedna od prednosti web-omogućenog LIMS-a je ta što sam server te baza podataka ne moraju biti na lokaciji laboratorija. Moguće je iznajmiti servere kao i baze podataka od strane trećih osoba čime se mogu smanjiti troškovi održavanja samog LIMS-a.
Dakako, u tom slučaju ostaje pitanje sigurnosti, no kako je poslovna logika odvojena od baze podataka te je zatvorenog oblika, čak i da dođe do sigurnost propusta bilo bi jako teško virtualnom napadaču odrediti smisao podataka.

Dakle, prednosti web-omogućenog LIMS-a su:

- Niži troškovi proizvodnje
- Proslijeđivanje troškova sigurnosti trećim strankama
- Lakšu obradu i dostavu rezultata analize podataka

Nedostaci su: (Ulma i Schlabach, 2005)

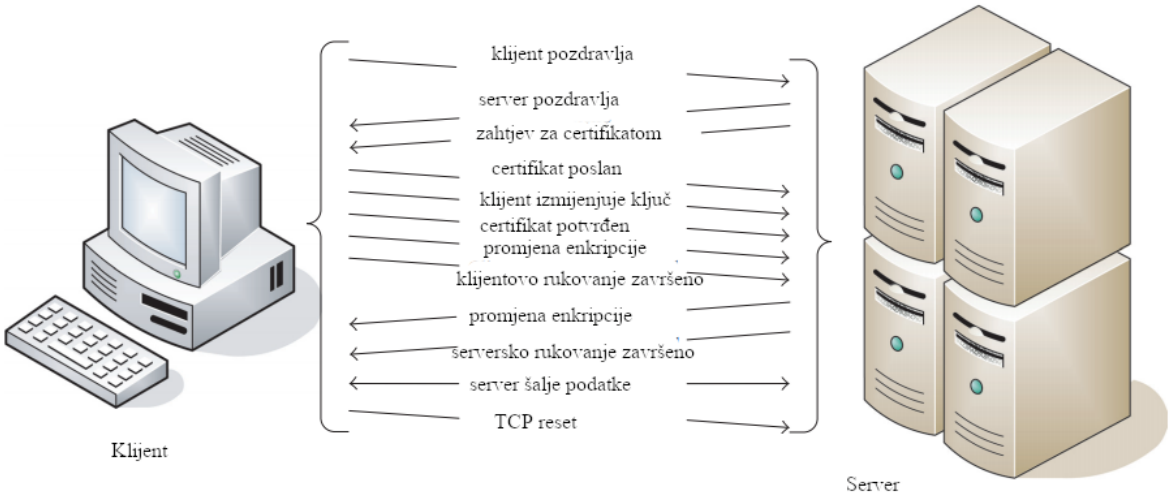
1. Sigurnost mreže
2. Istodobna obrada i integritet podataka
3. Osigurati resurse poslužitelja i ekonomičnost
4. Omogućiti distribuciju i kompatibilnost
5. Osigurati upotrebljivost

Metode otklanjanja potencijalnih nedostataka su: (Ulma i Schlabach, 2005)

1. Sigurnost mreže

Kad je sigurnost prijenosa podataka problem, najbolja opcija je korištenje SSL-a, tj. Secure Socket Layera preko TCP / IP[24] veza. SSL je protokol koji komunikaciju putem Interneta prenosi u šifriranom obliku. SSL osigurava da se informacije šalju, nepromijenjene, samo na poslužitelj na koji je informacija poslana.. Web stranice za kupnju putem interneta često koriste SSL tehnologiju za zaštitu podataka o kreditnoj kartici, a web stranice zdravstvene skrbi moraju koristiti SLL prilikom prijenosa medicinskih podataka putem Interneta. SSL sesije uvijek započnite s razmjenom poruka koje se nazivaju SSL "handshake" (stisak ruke). Stisak ruke omogućuje poslužitelju da autentificira klijenta

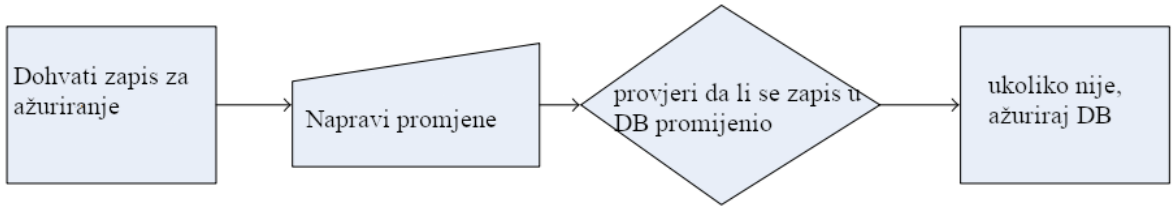
[24]TCP/IP - Transmission Control Protol/Internet Protocol je skup protokola kojim se definira metoda komunikacije uređaja preko Interneta



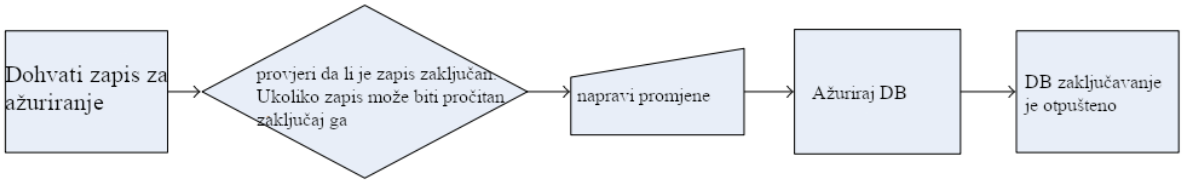
Slika 14. Prikaz SSL rukovanja između klijenta i servera (prilagođeno prema Ulma i Schlabach, 2005)

2. Istodobna obrada i integritet podataka

Jedini način sprečavanja istodobne obrade podataka je zaključavanje zapisa koji se uređuju. Postoje dva osnovna pristupa zaključavanju - optimistični i pesimistički. Ta dva pristupa se razlikuju po pretpostavci mogućnosti za događanje istovremene obrade podataka. Pesimistično zaključavanje zahtijeva da ostanete povezani s bazom podataka tokom cijelog postupka, dok optimistično pristup ne zaključava red tablice prilikom čitanja. Kad korisnik želi ažurirati red u bazi podataka, program mora utvrditi je li drugi korisnik promijenio redak otkako je redak pročitan.



Slika 15. Pregled optimističnog pristupa zaključavanja zapisa (prilagođeno prema Ulma i Schlabach, 2005)



Slika 16. Pregled pesimističkog pristupa zaključavanju podataka. Dok se vrši čitanje podataka zapis ostaje zaključan. (prilagođeno prema Ulma i Schlabach, 2005)

3. Osiguravanje resursa poslužitelja i ekonomičnost

Najbitniji elementi koje je potrebno imati da se zadrže visoke performanse aplikacije su odabir ispravnog modela klijent / poslužitelj, mehanizma za povezivanje baze podataka i optimistična metoda zaključavanja.

4. Omogućavanje distribucije i kompatibilnosti

HTML baziran front-end[25] ostvaren putem web preglednika je najbolja opcija kada treba uzeti u obzir kompatibilnost na više platformi. Pojednostavljeno je i održavanje web-bazirane aplikacije jer klijent koristi najnoviju verziju izravno s poslužitelja bez potrebe za instaliranjem softvera ili ponovne konfiguracije postavki.

5. Osiguravanje upotrebljivosti

Ljudi se bave najčešće koriste jednim ili samo par primjerom softvera za produktivnost dok svakodnevno obavljaju svoj posao. Stoga, ako se alat ne integrira dobro u trenutne prakse, onda može biti neiskorišten. Prilikom dizajna LIMS-a bitno je da je poželjan.

4.2. Odabir programskog jezika za serverski dio LIMS-a

Jedna od najbitnijih odluka u razvoju bilo kakvog računalnog procesa je odabir programskog jezika koji će se koristiti. Mada je većina high-level računalnih jezika[26] više-manje podjednaka u svojim mogućnostima, nastaju velika razmatranja kada se uzmu u obzir postojanje različitih jezičnih okvira koji su razvijeni za lakši i brži razvoj aplikacija. Također, između ostalog treba uzeti u obzir i veličinu programerske zajednice koja koristi

[25]front-end - ili prednji kraj, označava u web programiranju klijentsko sučelje

[26]high-level računalni jezici - jezici veće apstrakcije koji omogućavaju jednostavnije pisanje koda. U direktnoj su suprotnosti low-level jezici poput strojnog koda itd.

određeni programski jezik. Prilikom razvijanja softvera neizbježno je pojavljivanje bug[27]-ova (grešaka) u kodu, koji mogu biti lakše otklonjeni ukoliko je programerska zajednica veća. Nadasve, jedno od najvažnijih kriterija prilikom odabira programskog jezika je i brzina razvoja same aplikacije, pošto su troškovi razvoja ponajviše određeni troškom osoblja.

Python

Python je interpretirani, objektno orijentirani programski jezik visoke razine s dinamičnom semantikom. Podatkovne strukture su izgrađene u sam jezik. Dinamičko tipkanje i dinamičko povezivanje, čine ga vrlo atraktivnim za brz razvoj aplikacija, kao i za uporabu kao skriptni ili kao jezik za međusobno povezivanje postojećih komponenti. Python-ova jednostavna sintaksa koja se lako uči naglašava čitljivost i time smanjuje troškove održavanja programa. Python podržava module i pakete, što potiče modularnost programa i ponovnu upotrebu koda. Interpretator Python i opsežna standardna knjižnica dostupni su u izvornom ili binarnom obliku bez naplate za sve glavne platforme i mogu se slobodno distribuirati. (Oliphant, 2007)

Primjene Pythona: (https://www.python.org/)

- Desktop aplikacije utemeljene na GUI-u
- Aplikacije za obradu slika i grafički dizajn
- Znanstvene i računske primjene
- Igre
- Web okviri i web aplikacije
- Enterprise i poslovne aplikacije
- Operativni sustavi
- Razvoj jezika
- Izrada prototipova

Prednosti Pythona:

Raznovrsna primjena jezika Python rezultat je kombinacije značajki koje ovom jeziku daju prednost nad drugima. Neke od prednosti programiranja u Pythonu uključuju:

1.Mogućnost proširivanja modula

[27]bug - direktan prijevod buba, označava greške u kodu. Naziv je dobio prema činjenici da je u prvim računalnima grešku upravo uzrokovali kukci koji su uzrokovali kratke spojeve u tadašnjim računalima koji su bili veličine cijele prostorije

Jedna od najvećih prednosti upotrebe Pythona u znanstveno računalnom okruženje je njegova velika mogućnost proširivanja modula. Moduli se mogu ugraditi za dodavanje novih funkcija i mogućnosti interpretatora Pythona. Zajednica Python jezika (https://wiki.python.org/) jedan je od sjajnih izvora gdje se mogu pronaći korisni moduli. Većina dostupnih modula su pod licencom otvorenog koda koje omogućuju prilagodbu za posebne aplikacije. U znanosti jedne od najbitnijih knjižnica napravljenih u Python-u su:

A. SciPy

SciPy (http://scipy.org) znanstvena je računalna knjižnica koja se može koristiti u mnoštvu različitih konteksta. SciPy dolazi s bitnim funkcijama obrade signala, npr. savijanje, filtriranje, spektralna analiza i sl.

B. NumPy

NumPy (http://www.numpy.org/) je numerička knjižnica koja pruža korisne strukture podataka, kao što su višedimenzionalni nizovi i matrice, zajedno s njihovim matematičko-operacijskim rutinama. NumPy vrši razne manipulacije, kao npr. zbrajanje, oduzimanje, množenje prikladnijim i bržim nego isključivo korištenje primitivne vrste podataka Pythona. NumPy također sadrži neke korisne matematičke funkcije koje se obično koriste u obradi signala, npr. prosjek, trigonometrija, skraćenje itd.

C. Matplotlib

Matplotlib (http://matplotlib.org/) je knjižnica za vizualizaciju podataka. Matplotlib se može koristiti za stvaranje mnogih vrsta parcela, npr. dvodimenzionalnih grafikona, trodimenzionalnih grafikona, spektrograma, histograma itd. Također može pokazati legendu, označavanje osi i bojanje grafikona.

2. Velika standardna biblioteka

Python nudi veliku standardnu biblioteku koja uključuje područja poput internetskih protokola, string operacija, alata za web usluge i sučelja operacijskog sustava. Mnogo zadataka programiranja visoke uporabe već su skriptirani u standardnu knjižnicu što znatno smanjuje dužinu koda koji se mora pisati. (https://docs.python.org)

3. Otvoreni izvor i razvoj zajednice:

Jezik Python razvijen je pod licencom otvorenog koda odobrenog od OSI[28], što ga čini slobodnim za korištenje i distribuciju, uključujući u komercijalne svrhe.
Nadalje, njegovim razvojem upravlja zajednica koja surađuje na kodu putem konferencija. (https://www.python.org)

4. Jednostavnost učenja i dostupnost podrške:
Python nudi izvrsnu čitljivost i nepreglednu sintaksu za jednostavno učenje, što pomaže početnicima da koriste ovaj programski jezik. Smjernice za stil koda, PEP 8 (https://www.python.org), pružaju skup pravila koja olakšavaju oblikovanje koda. Uz to, široka baza korisnika i aktivnih programera rezultirala je bogatom bankom sredstava internetskih resursa za poticanje razvoja i daljnjeg usvajanja jezika.

5. Brze strukture podataka:
Python ima ugrađene strukture podataka kao što su liste, rječnici te hash tablice koje se mogu koristiti brzu obradu podataka. Nadalje, Python nudi i mogućnost dinamičkog tipkanja na visokoj razini. (https://www.python.org)

6. Produktivnost i brzina:
Python ima čist objektno orijentiran dizajn, pruža poboljšane mogućnosti upravljanja procesima i posjeduje snažne mogućnosti integriranja i obrade teksta, a svi te doprinosi povećanju njegove brzinu i produktivnost. Python se također smatra održivom opcijom za izgradnju složenih mrežnih aplikacija sa više protokola. (Oliphant, 2017)

[28]OSI - Open Systems Interconnection, tj. sustav otvorene međupovezanosti označava konceptualni model koji karakterizira standardne komunikacije modele kompjuterskog sistema

Tablica 5. Primjer stvaranja korisnika i postavljanje lozinke u Python-u. Korišten je framework Django. Prednost Python-a je u samo-objašnjivom kodu koji se lako čita

```
class UserSerializer(serializers.ModelSerializer):
    """serijalizator user objekta"""

    class Meta:
        model = get_user_model()
        fields = ('email', 'password', 'name')
        extra_kwargs = {'password': {'write_only': True, 'min_length': 5}}

    def create(self, validated_data):
        """kreairanje novog korisnika s enkriptiranom šifrom I vraćanje ga"""
        return get_user_model().objects.create_user(**validated_data)

    def update(self, instance, validated_data):
        """ažuriranje korisnika, postavljanje lozinke te vraćanje korisnika""
        password = validated_data.pop('password', None)
        user  = super().update(instance, validated_data)

        if password:
            user.set_password(password)
            user.save()
        return user
```

Ograničenja ili nedostaci Pythona:

Python ima mnoštvo povoljnih značajki, a i programeri preferiraju ovaj jezik nad drugim programskim jezicima jer ga je lako naučiti i kodirati. Međutim, ovaj jezik još uvijek nije zauzeo svoje mjesto u nekim računalnim arenama Stoga ovaj jezik možda neće biti korišten u nekim poduzećima, a ograničenja Python-a uključuju (Zadka, 2019):

1. Poteškoće u korištenju drugih jezika

Ljubitelji Pythona toliko su se navikli na njegove karakteristike i veliku knjižnicu, pa se suočavaju s problemom u učenju ili radu na drugim programskim jezicima.

2. Slab u mobilnom računarstvu

Python je bio prisutan na mnogim platformama radnih površina i poslužitelja, ali se smatra slabim jezikom za mobilno računalstvo. To je razlog zbog čega je u nju ugrađeno vrlo malo mobilnih aplikacija.

3. Sporiji od nekih jezika

Python se izvršava uz pomoć prevoditelja, što ga usporava, jer kompajliranje pomaže u brzini rada. S druge strane, dovoljno je brz i za mnoge web aplikacije.

4. Pogreške tijekom rada

Jezik Python dinamički se tipka tako da ima mnoštvo ograničenja dizajna, neke od njih su uočavanje grešaka u kodu. Čak se vidi da to zahtijeva više vremena za testiranje, a pogreške se pojavljuju kada se aplikacije napokon pokrenu.

PHP

PHP je skriptni jezik na strani poslužitelja koji se koristi za izradu statičkih web stranica, dinamičnih web stranica ili web aplikacija. PHP označava Pretprocesor Hypertext, a ranije je označavao Personal Home Pages.
PHP skripte mogu se interpretirati samo na poslužitelju na kojem je instaliran PHP.
Računala klijenta koja pristupaju PHP skriptama trebaju samo web preglednik.
(https://www.php.net)

Prednosti PHP-a:

PHP sadrži jezične značajke skriptnog jezika kao i OOP (objektno orijentiranog programiranja). To znači da programeri iz različitih pozadina programskog jezika mogu naučiti ovaj jezik u kratkom vremenu.

Neke od prednosti PHP-a su:

1. Lako stvaranje skripti na razini poslužitelja

PHP je primarno usredotočen na mogućnosti skriptiranja na strani poslužitelja te pruža programerima potrebne alate za stvaranje web aplikacija koje su snažno dinamične prirode. Ima široku lepezu funkcionalnosti, npr. prikupljanje podataka iz obrasca, generiranje dinamičnih web stranica i postavki te pristup kolačićima radi kontrole sesije.

2. Može se koristiti na svim glavnim operacijskim sustavima

To uključuje UNIX i Linux i, dakle, web programerima daje izbor operativnog sustava i web poslužitelja prilikom implementacije aplikacije.

3. Sposobnost podržavanja različitih baza podataka.

Koristeći PHP web stranica s bazom podataka može se napisati s nevjerojatnom jednostavnošću koristeći bilo koju bazu podataka.

Može se koristiti s velikim brojem sustava za upravljanje relacijskim bazama podataka, radi na svim web preglednicima (primjerice: Apache, osobni web poslužitelj, Microsoft IIS, Netscape, iPlanet) i svim bazama podataka (primjer: MySQL, dBase, IBM DB2, ODBC, PostgreSQL, Inter Base, Front Base, SQLite). PHP5 potpuno je objektno orijentiran jezik koji se može postaviti gotovo svugdje. Također ima izvrsnu dokumentaciju. (https://www.php.net)

Nedostaci PHP-a:

1.Sigurnost:

S obzirom da je izvorni kod otvoren, svi ljudi mogu vidjeti izvorni kod. Ako u izvornom kodu postoje greške, ljudi ga mogu koristiti za iskorištavanje slabosti.

2.Nije pogodan za velike aplikacije:

Bit će ga teško koristiti za programiranje ogromnih aplikacija. Kako programski jezik nije izrazito modularan, ogromne aplikacije stvorene iz programskog jezika teško će se održavati. (https://github.com/)

3. Slabo tipkanje:

Implicitna konverzija može iznenaditi programere i dovesti do neočekivanih pogrešaka. Konfuzija između nizova i hash tablica.. Često postoji nekoliko načina za izvršavanje zadatka. Jezik nije snažno tipkan što povećava mogućnost pogrešaka. (Becht i Staples, 1995)

Ruby

Ruby je dinamičan, reflektivni, objektno orijentirani programski jezik opće namjene. Ruby je čisti objektni jezik koji je razvio Yukihiro Matsumoto. Sve u Ruby-u je objekt osim blokova, ali i za njega postoje zamjene, tj. Procs i lambda. Cilj Ruby-ovog razvoja bio je omogućiti da djeluje kao razuman posrednik između ljudskih programera i osnovnih računalnih strojeva. **(**https://www.ruby-lang.org)

Ruby on Rails: Rails je razvojni okvir koji web programerima daje strukturni okvir za sav kôd koji pišu. Framework Rails pomaže programerima u izradi web stranica i aplikacija apstrahiranjem i pojednostavljivanjem većine zadataka koji se ponavljaju. (https://rubyonrails.org/)

Prednosti Ruby-a (Holzner i Wrox, 2007):

1. Prilagodljiv jezik:

Mape i metode ubrizgavanja omogućuju sastavljanje koda u kompaktan i čitljiv način u usporedbi s konvencionalnim načinom omotavanja koda unutar petlje

2. Lakoća obrade teksta

String[29]-ovi u Ruby-u su promjenjivi i pružaju izuzetno sveobuhvatni API. Kompaktnija sintaksa i još neke klase omogućavaju obavljanje brojnih zadataka vrlo brzo u Ruby-u nego na Javi.

3. Mala količina koda za veliku funkcionalnost

Manje koda također čini kod lakšim za održavati.

```
def lock
  while (Thread.critical = true; @locked)
    @waiting.push Thread.current
    Thread.stop
  end
  @locked = true
  Thread.critical = false
  self
end
```

Slika 17. Primjer sintakse u Ruby-u. Ovaj kod se baci kontrolom istovremenosti procesa. (prilagođeno prema Sheehan, 2007)

Nedostatci Ruby-a: (Sheehan, 2007)

1. Nijedan operacijski sustav nije implementiran u Ruby-u

[29]String - tj. "niz" je jedan od najčešće definiranih primitivnih oblika podataka u programerskim jezicima. Označava niz znakova.

Ta činjenica otežava samo učenje programskog jezika Ruby. Korištenje Ruby-a na Windowsima je teže pošto Ruby prikazuje svoje Unix korijene sasvim jasno. Datoteka i postupak naredbe za manipulaciju proizlaze iz Unix modela.

2. Opći nedostatak Ruby dokumentacije na engleskom jeziku

Jedna od najvećih primjedbi Ruby-u je što je napisano malo knjiga te ima jako malo programskih resursa dostupnih na engleskom jeziku. Ta činjenica otežava postupak čišćenja koda te produžuje vrijeme stvaranja računalnog programa.

.

Java

Java programski jezik općenit, istovremen, snažno tipkani, objektno usmjereni jezik. Obično se sastavlja u skupu naredbi za bajt kod i binarnom formatu definiranom u Specifikaciji Java virtualnog stroja. (https://docs.oracle.com/)

Prednosti korištenja Jave: (Sarmenta, 1998)

1. Jednostavnost upotrebe

U sustavu koji pokreće Java-u, potrebno je samo koristiti preglednik koji podržava Java za posjetu web mjestu poslužitelja. Nema tehničkog znanja koje je potrebno izvan upotrebe web preglednika. Dakle, bilo tko, pa makar i minimalno računalno pismeni korisnici internetskih usluga, mogu koristiti Java aplikaciju

2. Neovisnost o platformi

Sustavi temeljeni na Javi su jednostavni za korištenje programerima i administratorima sustava. Budući da su jezik Java-e kao i knjižnice neovisni o platformi, programeri mogu „pisati jednom, pokrenuti bilo gdje“. Što znači da programeri mogu raditi svoj kod na bilo kojoj platformi, sastaviti konačni kod u jedan skup binarnih datoteka (tj. Java bytecode), a zatim je postaviti na odgovarajuće web stranice. Zatim, svaki stroj koji ima Java runtime[30] okruženje može preuzeti te aplete[31] i pokrenuti ih.

3. Sigurnost.

[30]Java runtime - označava kompjutersko okruženje koje pokreće Java virtualni stroj te time omogućava izvršavanje i pokretanje aplikacija napisanih u Javi

[31]aplet - označava malu aplikaciju (engl. applet)

Poput raširene mreže, sustav temeljen na Javi ima prednost što ne zahtijeva račune ili pristup udaljenim shell[32]-ovima na korisničkim računalima. Nadalje, za razliku od distribuiranih mreža, Java apleti koje korisnici preuzimaju izvršavaju se u kodnom pješčaniku koji im sprečava krađu ili činjenje podatkovne štete

Nedostaci Jave: (Gibbons, 1998)

1. Programi mogu biti jako verbozni

Obzirom da je Java objektno-orijentirani jezik, dužina koda se može smatrati kao i nužnom posljedicom.

2. Programski kompleksan unos tipkovnicom

Na primjer za učitavanje cjelobrojne vrijednosti iz tipkovnice, prvo se mora prijaviti input stream (struja unosa) te se ona zatim pričvrsti na tipkovnicu. Štoviše, Java je vrlo oprezna po pitanju iznimaka; ovo je općenito dobra činjenica, no to znači da bilo koja metoda koji izvodi ulaz mora definirati IOException.

```
import java.io.*;

public class PS {

   public static void main(String[] args) {
      Runtime runtime = Runtime.getRuntime();
      try {
         Process ps_process =
            runtime.exec("ps x");
         BufferedReader in = new BufferedReader(
            new InputStreamReader(
               ps_process.getInputStream()));
         String line;
         while ((line = in.readLine()) != null)
            doSomething(line);
      }
      catch (IOException e);
   }

}
```

[32]shell - prijevod "ljuska", u UNIX sustavima označava interaktivno korisničko sučelje preko kojeg se vrši komunikacija sa računalnim sustavom

Slika 18. Primjer verboznosti Java sinktakse. Ovaj kod je zadužen samo za čitanje unosa tipkovnicom (prilagođeno prema Sheehan, 2007)

3. Pomalo neobična sintaksa

Java je naslijedila sintaksu od programskog jezika C. Zagrade oko izjava if i else nisu potrebni, barem za sprječavanje nejasnoća. Nesretno je za što se koristi '=' za dodjeljivanje, a ne za definiranje jednakosti. Možda je najgore od svega sintaktička nejasnoća oko if-else izjava.

Javascript

JavaScript je skriptni ili programski jezik koji omogućuje implementaciju složenih radnji na web stranicama - svaki put kada web stranica učita više neće samo prikazivati statičke informacije, već Javascript omogućuje prikazivanje pravovremenih ažuriranja sadržaja, interaktivnih karata, animiranih 2D / 3D grafika i sl. (https://developer.mozilla.org)

Mada je Javascript izumljen primarno za kontrolu web preglednika, izumom Node.js-a Javascript se također preselio i na web servere.

Node.js

Kao asinhrono izvršavajući JavaScript runtime[33], Node.js dizajniran je za izradu skalabilnih mrežnih aplikacija. Mnogim se vezama može raditi istovremeno. Nakon svake veze, povratni poziv se pokreće, ali ako nema posla, Node.js će se isključiti. (https://nodejs.org)

Prednosti Javascripta: (Hughes-Croucher & Wilson, 2012)

1. Zbog razvoja Node.js, može se koristiti i na poslužitelju i na klijentu

Programer koji radi i na serveru i na klijentu na kraju se neće morati prebacivati između dva jezika. Prebacivanje između različitih zadataka ili jezika smatra se velikim troškom za programera i može smanjiti produktivnost. Ponovna upotreba koda koji se izvodi i na strani poslužitelja i na strani klijenta je moguć kada je jezik na obje strane isti.

[33]Javascript runtime - kompjutersko okruženje koje pokreće Javascript motor

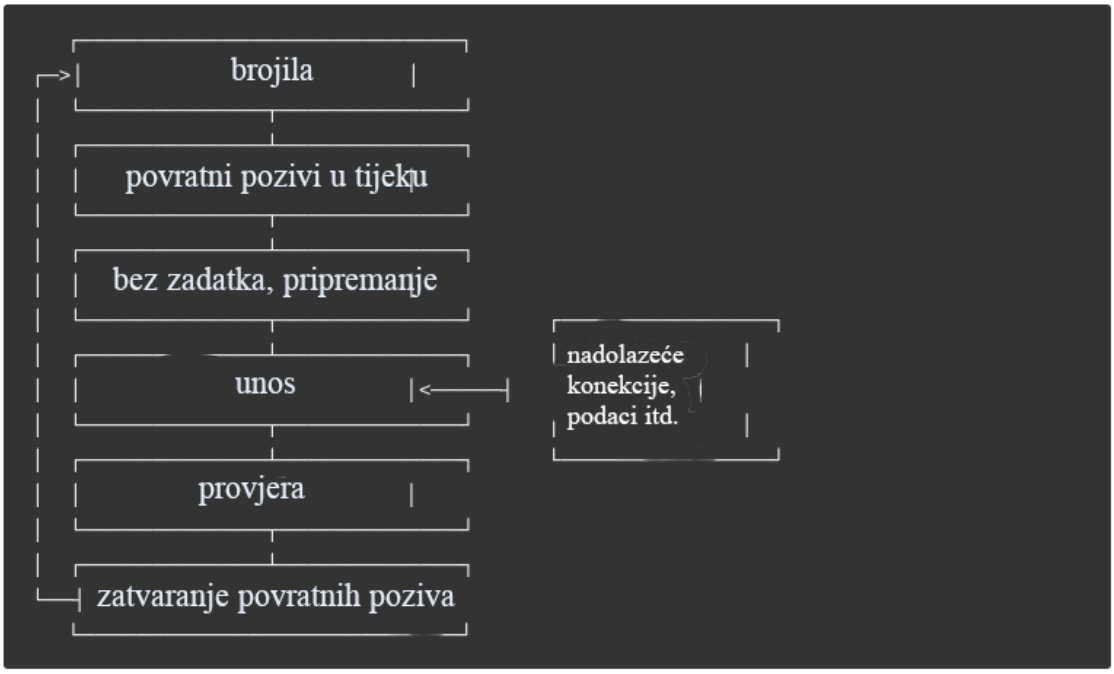
Tablica 6. Primjer jednostavnog postavljanja servera u Node.js-u. Server je postavljen na lokalnom računalu na portu 3000. Cijela komunikacija se odvija putem zahtjev/odgovor metodologije.

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Pozdrav svijete!');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

2. Node.js ima ugrađen event loop[34]

Svi dostupni moduli i knjižnice ga koriste. Zbog toga je rad s više dijelova u asinhronom programu puno lakši .

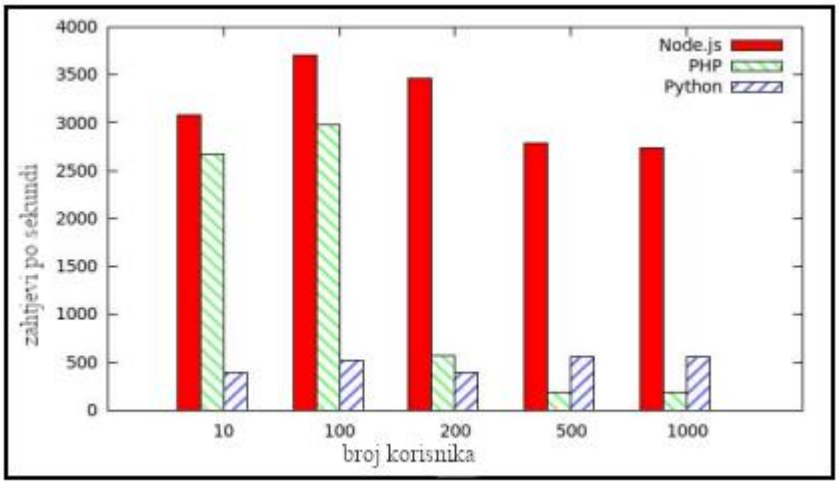
[34]event loop - "petlja događaja" je osnovni koncept Node.js koji je preuzet od načina na koji JavaScript funkcionira u web pregledniku. Petlja događaja je zadužena za obradu asinhronih događaja bez da se blokira glavni proces



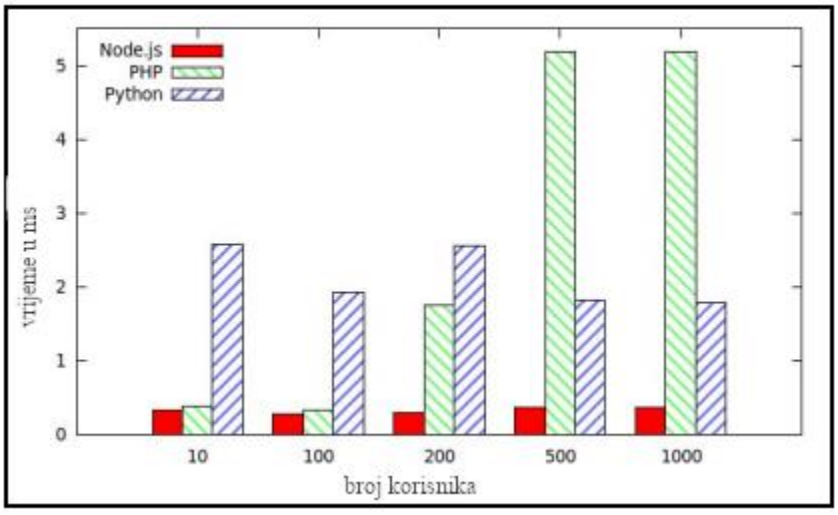
Slika 19. Pregled Node.js event loop-a. Za razliku od većine drugih jezika, Node.js se obrađuje na samo jednom procesu, a event loop omogućava jednostavno upravljanje asinhronim akcijama (prilagođeno prema www.nodejs.org pristupljeno 26.3.2020)

3. Izvrstan je za procese koji zahtijevaju visok Input/Output

Ukratko, Node.js djeluje znatno bolje od PHP-a u situaciji visoke istovremenosti zahtjeva, PHP obrađuje male zahtjeve dobro, ali bori se sa velikim zahtjevima. Osim toga, Node.js preferira se koristiti u IO-intenzivnim aplikacijama. (Lei i sur., 2014)



Slika 20. Broj zahtjeva po sekundi koji se može obraditi ovisno o broju korisnika. Usporedba između Node.js-a, PHP-a i Pythona-a. Vidljivo je da s povećavanjem broja korisnika performanse PHP-a znatno opadaju. Python-ove mogućnosti su konstantne, no ipak daleko manje usporedno sa Node-js-om. Treba napomenuti da je u pitanju jednostavno slanje zahtjeva prema serveru. (prilagođeno prema Lei i sur., 2014)



Slika 21. Analiza vremena potrebnog za obradu korisničkog zahtjeva u Node.js-u, PHP-u te Python-u. vidljivo je kako performanse PHP-a znatno opadaju brojem korisnika, dok Python nema takvih poteškoća sa skaliranjem. Doduše Node.js najbrže obrađuje zahtjeve od sva tri jezika. (prilagođeno prema Lei i sur., 2014)

Nedostaci Javasripta:

1. Node.js je jako loš u aplikacijama koji su CPU intezivni

Upravo zbog same arhitekture blokirajućeg event loop-a, Node.js je jako spor pri obradi zahtjeva koji su jako CPU intezivni. To znači da sama aplikacija se zaustavlja za sve korisnike sve dok se zahtjev ne izvrši. (Lei i sur., 2014)

2. Asinkrono programiranje je novina za mnoge programere

Asinkrono programiranje pogonjeno event-ovima novi je koncept za mnoge programere te može potrajati dok se programer ne navikne. To može utjecati na produktivnost. Također, otklanjanje pogrešaka postaje teško jer stack trace[35]-ovi više ne predstavljaju kontrolni tijek za obradu određenog zadatka. (Hughes-Croucher i Wilson, 2012)

Konačan odabir programskog jezika

Uzevši u obzir najčešće korištene programske jezike koje se koriste na serverskom dijelu web aplikacije, potrebno je napraviti analizu potrebnih stavki pomoću kojih se može odrediti najkorisniji programski jezik.

Oni su:

- Lakoća pisanja
- Otvorenost koda
- Veličina programerske zajednice
- Dostupnost literature
- Raširenost korištenja u industriji
- Dostupnost i razvijenost programskog okvira koji ubrzavaju razvoj aplikacije
- Brzina izvršavanja koda

Uzevši u obzir navedene stavke, izbor se sužava na programske jezike Python, Ruby te Javascript (Node.js).

U usporedbi Python-a te Node.js-a, Python ima mnogo zrelih okvira za razvijanje web stranice velikih razmjera, poput YouTubea i Source Forge-a. Node.js je tehnologija u nastajanju i ima brojne prednosti u sustavu IO-intenzivnih situacija, ali malo je teška za programere koji nisu upoznat sa asinkronim programiranjem. (Lei i sur., 2014) Također,

[35]stack trace - pratitelj stoga, označava sve aktivne stogove koje se koriste u izvršavanju programa

obzirom da je Node.js jako loš u obradi zahtjeva koji su jako intezivni za CPU-u, Node.js neće moći biti korišten u LIMS-u koji obrađuje velike količine podataka za potrebe izvještavanja i sl.

Odluka ostaje između Pythona-a te Ruby-a, no obzirom da je Python iznimno raširen u znanstvenoj zajednici te obzirom da je razvijeno mnogo modula koji su praktički nezamjenjivi u obradi velike količine podataka upravo u Python-u, kao i činjenici da je programerska zajednica daleko veća od Ruby-a, Python se pokazuje kao najbolji odabir.

4.2.1. Odabir frameworka za izradu serverskog dijela aplikacije

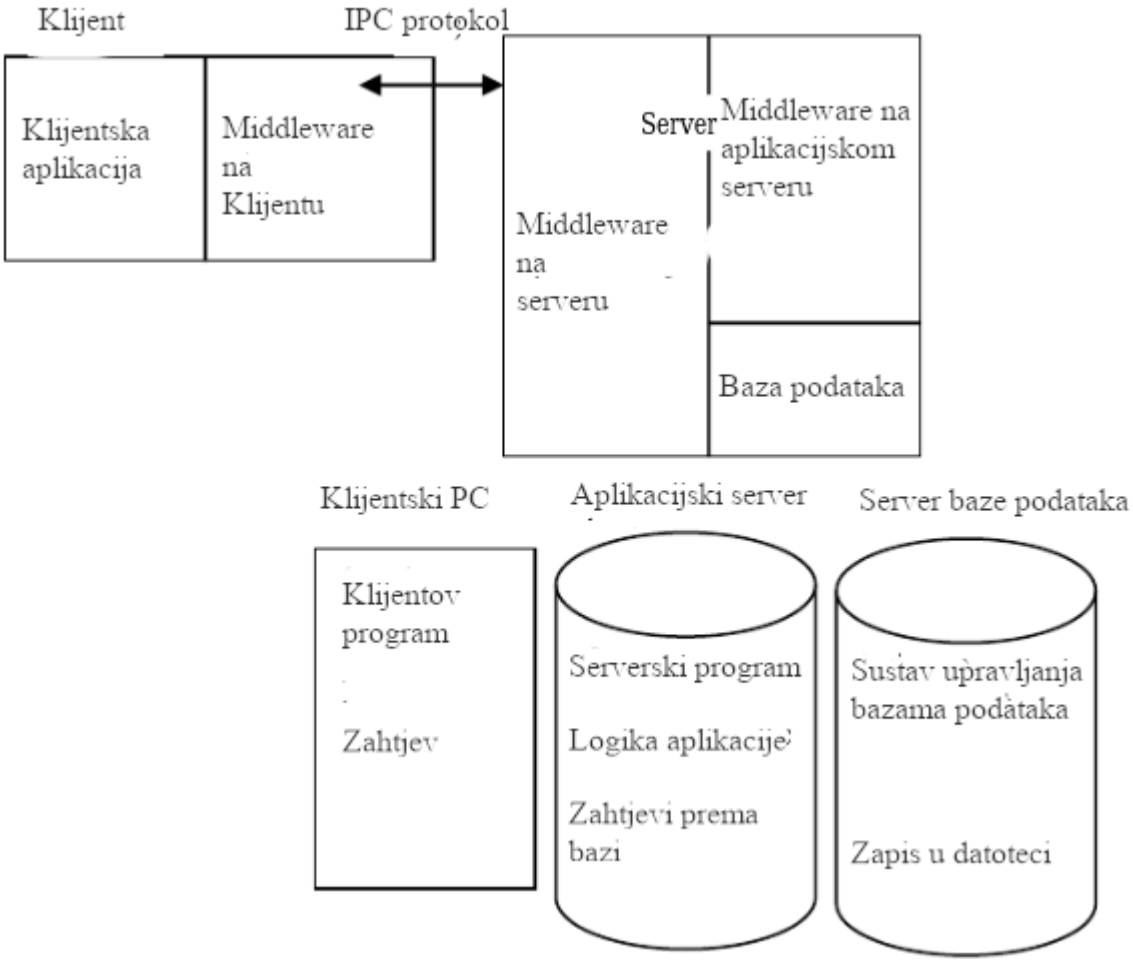
Nakon što je programski jezik Python odabran za izradu serverskog dijela aplikacije, nužno je odabrati framework (programski okvir) pomoću kojeg će se aplikacija kodirati.
Neke od prednosti frameworka su:

- Veća brzina izrade aplikacije
- Lakše testiranje koda
- Podrška za lakše otklanjanje grešaka u kodu
- Uklanjanje potrebe za repetitivnim kodom

Tokom godina razvijeno je mnogo frameworka u Python-u, a 3 najpopularnija frameworka na programerskoj stranici Github su:

1. Flask
2. Django
3. Tornado

Prilikom odabira frameworka valja uzeti u obzir programersko sučelje kojim je definirana komunikacija između samog servera (poslužitelja) te frameworka. Komunikacija između servera te web aplikacije se odvija putem petlje zahtjeva i odgovora, te se u 3-slojnoj poslužitelj-klijent arhitekturi odvija putem posredničkog software-a.



Slika 22. Pregled 3-slojne arhitekture server-middleware-klijent. Middleware je posrednički softver koji dodatno obrađuje klijentove zahtjeva prije prosljeđivanja serveru. Također, vidimo i aplikacijski server koji može obrađivati složeni korisničke zahtjeva poput obrade podataka, dok je server baze podataka zaslužan za spremanje podataka te njihovo čitanje i posluživanje. (prilagođeno prema Haroon, 2014)

U suštini, klijent šalje zahtjev poslužitelju, poslužitelj pomoću posredničkog softvera prosljeđuje zahtjev web aplikaciji koja zahtjev obrađuje te obrađeni zahtjev šalje natrag klijentu u obliku odgovora. (Malaysia i Oluwatosin, 2014)

Python je definirao 2 specifikacije za prosljeđivanje zahtjeva od samih poslužitelja do aplikacija napisanih u Python-u, a one su:

1. WSGI
2. ASGI

WSGI

WSGI označava Web Server Gateway Interface (Web Poslužitelj Ulazno Sučelje) te je specifikacija koja definira način na koji poslužitelj komunicira sa web aplikacijama te kako više web aplikacija može biti međusobno povezano da bi obradile jedan zahtjev.
WSGI je detaljno opisan u specifikaciji PEP 3333. (https://www.python.org)

WSGI sučelje ima dvije strane: stranu poslužitelja i stranu aplikacije. Strana poslužitelja poziva objekt koji se može instancirati te kojeg pruža strana aplikacije. Specifičnosti pružanja tog objekta ovisi o poslužitelju. Neki poslužitelji mogu zahtjevati od implementatora aplikacije da napiše kratku skriptu kako bi stvorio instancu poslužitelja i opskrbio je objektom aplikacije. Ostali poslužitelji i pristupnici mogu koristiti konfiguracijske datoteke ili druge mehanizme za određivanje odakle bi trebalo uvesti objekt aplikacije.

Pored "čistih" poslužitelja i aplikacija, također je moguće stvoriti posredničke komponente koje implementiraju obje strane ove specifikacije. Takve komponente djeluju kao aplikacija na poslužitelju te kao server na aplikaciji i mogu se koristiti za pružanje proširenih API-ja, transformacije sadržaja, navigacije i drugih korisnih funkcija. (https://www.python.org)

Bitna stavka je da je WSGI sučelje sinkrono, što znači da se svaki zahtjev odrađuje jedan po jedan. Doduše, poslužitelji mogu odrađivati više zahtjeva odjednom stvarajući odvojene procese za svaki zahtjev, no ukoliko dolazi do mnogo zahtjeva odjednom to može zauzeti jako mnogo resursa poslužitelja. (https://www.python.org)

ASGI

ASGI je skraćenica za Asynchronous Server Gateway Interface (Asinkroni Poslužitelj Ulaznog Sučelja) te je nasljednik WSGI-a, a namijenjen je pružanju standardnog sučelja između web-poslužitelja, frameworka i raznih aplikacija.

Tamo gdje je WSGI pružio standard za sinkrone aplikacije Python, ASGI nudi mogućnost i za asinhrone i za sinkrone aplikaciju, s time da je unazad kompatibilan s WSGI-em. (https://asgi.readthedocs.io).
Asinkronost označava pojam da je moguće obraditi više zahtjeva odjednom.

Svrha ASGI-a je pružiti standardno sučelje između poslužitelja mrežnih protokola (posebno web poslužitelja) i Python aplikacija te je namijenjeno omogućavanju više uobičajenih stilova protokola (uključujući HTTP, HTTP/2[36] i WebSocket[37]).

ASGI se sastoji od dvije različite komponente (https://asgi.readthedocs.io):

- Poslužitelja protokola
- Aplikacije koja se nalazi unutar protokolarnog poslužitelja te se instancira jednom po vezi i upravlja porukama događaja

Za razliku od WSGI-a, ASGI veza sadrži također:

- Opseg veze, koji predstavlja protokolarnu vezu s korisnikom i opstaje dok se veza ne prekine.
- Događaje koji se šalju aplikaciji kako se stvari događaji na vezi

Prednosti ASGI-a nad WSGI-em su:

- Bolje performanse
- Podrška za WebSocket te HTTP/2 protokol
- Podrška za pozadinske zadatke

Uzevši u obzir sve navedeno ASGI specifikacija je superiornija WSGI-u te se predlaže da bude implementirana na poslužitelju.

Flask

Flask je mikroframework Pythona koji pruža osnovne funkcionalnosti web frameworka i dopušta više nadogradnji koji se dodaju kako bi se funkcionalnost i skup mogućnosti mogli proširiti (https://flask.palletsprojects.com) . Flask se naziva mikro okvir Pythona jer čini jezgru funkcionalnosti jednostavnom, ali proširivom. (Maia, 2015)
Flask koristi Jinja Template Engine i Werkzeug WSGI Toolkit.
Prednosti korištenja Flaska su (Maia, 2015):

[36]HTTP/2 - Označava nadogradnju HTTP protokola. Omogućava dvostranu komunikaciju između klijenta i poslužitelja bez konstantne razmjene zahtjeva i odgovora. Omogućava konstantnu komunikaciju preko "steam-ova", tj. struje podataka

[37]Websocket - kompjuterski komunikacijski protokol koji omogućava punu duplex vezu preko jedne TCP konekcije

- Modularnost:

Flask ima lagan i modularan dizajn, tako da ga je lako transformirati dodavajući proširenje po potrebi, bez da bude glomazan

- ORM-agnostičan:

Može se priključiti na bilo kojii ORM[38] (Object-relational Mapper) kao npr. SQLAlchemy (https://www.sqlalchemy.org/). ORM omogućava lakšu komunikaciju s bazom podataka koristeći programerski jezik izvora, izbjegavajući komplicirane SQL upite

- Osnovni API je lijepo oblikovan i koherentan

Osnovna dokumentacija je opsežna, prepuna primjera i dobro strukturirana.

- Jednostavan za upotrebu:

Vrlo je jednostavno napraviti web poslužitelj spreman za korištenja, a Flask je kompatibilan sa WSGI 1.0 specifikacijom

- Visoka fleksibilnost

Konfiguracija je iznimno fleksibilna, pružajući obilje rješenja za sve potrebe proizvodnje

Tablica 7. Primjer jednostavnosti postavljanja Flask servera. Definiranje ruta se postiže preko dekoratora. U ovom primjeru se vidi definiranje ruta i subruta.

```
from flask import Flask
from markupsafe import escape

app = Flask(__name__)

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

@app.route('/user/<username>')
def show_user_profile(username):
    # pokaži korisnički profil za tog korisnika
    return 'User %s' % escape(username)

@app.route('/path/<path:subpath>')
def show_subpath(subpath):
    # pokaži podrutu poslije /path
```

[38]ORM - Object Relation Mapper ili maper odnosa objekata, je programska tehnika koja omogućava komunikaciju s međusobno nekompatabilnim sistemima koristeći objektni programerski jezik. U ovom kontekstu omogućava se upravljanje bazama podataka preko programskog jezika kao što je npr. Python bez pisanja SQL upita

Django

Django je framework napisan u Pythonu koji potiče brzi razvoj i čist, pragmatičan dizajn. (https://www.djangoproject.com/)

Ono što čini Django jednim od najpopularnijih frameworka je: (Pinkham i Sams, 2016)

1. Ubrzava razvoj web aplikacija

Django je jedan od najzrelijih frameworka za Python. Njegova pravila dizajna usredotočena su na znatno smanjenje vremena za razvoj web aplikacija. Značajke koje pruža Django omogućuju programerima da brzo izrađuju web aplikacije u skladu s različitim poslovnim zahtjevima. Veliki postotak Python programera se odlučuje za Django kada moraju ispuniti ciljeve i rokove.

2. Zamišljen je kao framework s potpunom funkcionalnošću

Tijekom razvijanja web aplikacije Django pruža programe koji su potrebni programerima direktno u samom frameworku bez potrebe za instaliranjem raznih modula. Također pruža kôd za uobičajene operacije poput manipulacije bazama podataka, HTML predloške, usmjeravanja URL-ova, upravljanja sesijama i sigurnosti. Sve navedeno pomaže programerima da znatno skrate vrijeme razvoja web aplikacija.

3. Podržava MVC paradigmu programiranja

MVC paradigma programiranja omogućava programerima da odvoje korisničko sučelje i slojeve poslovne logike. Pristup nadalje pomaže programerima da pojednostave i ubrzaju razvoj velikih web aplikacija odvajajući slojeve kompleksnosti. Django nadalje omogućava programerima da ponovo upotrebljavaju istu poslovnu logiku u više projekata.

4. Kompatibilan je s glavnim operativnim sustavima i bazama podataka

Danas korisnici pristupaju web aplikacijama na raznim uređajima i platformama. Django povećava pristupačnost web aplikacija podržavajući operativne sustave poput Windows, Linux i MacOS-a, ORM sustav koji pruža Django olakšava programerima rad s nekoliko široko korištenih baza podataka. Oni čak mogu koristiti ORM sustav za obavljanje uobičajenih operacija baze podataka i migriranje iz jedne baze u drugu bez pisanja dodatnog koda.

5. Pruža snažne sigurnosne značajke

Ugrađene sigurnosne značajke koje pruža Django pomažu programerima da zaštite web aplikacije od različitih ciljanih sigurnosnih napada, ubrizgavanje SQL-a i krivotvorenje zahtjeva na više stranica. Istovremeno, povećava sigurnost web aplikacija sprječavajući uobičajene sigurnosne pogreške vezane uz kodiranje Python-om.

6. Lako se proširuje

Django se kontinuirano razvija kako bi programerima omogućio izgradnju boljih i modernijih web aplikacija. U isto vrijeme, razvojni programeri Django mogu lako prilagoditi i proširiti web framework unoseći promjene u svoje nevezane komponente. Django ima mogućnost iskopčavanja ili zamjene tih razdvojenih komponenata u skladu s preciznim zahtjevima pojedinih projekata.

7. Podržan od strane velike i aktivne zajednice

Kao framework otvorenog koda, Django pomaže programerima da značajno smanje troškove razvoja web aplikacija. No, to podržava velika i aktivna zajednica programera. Članovi zajednice Django redovito ažuriraju nove dodatke i isječke koda kako bi pojednostavili razvoj web aplikacija. Programeri mogu lako ubrzati izradu prilagođenih web aplikacija koristeći prednosti ovih resursa koje su prenijeli članovi Django zajednice. Članovi zajednice čak pomažu programerima da riješe zajedničke probleme i probleme oko razvoja web aplikacija.

8. Podržava ASGI specifikaciju

Od verzije 3, Django je nadograđen tako da podržava ASGI specifikaciju što čini sami framework asinhronim. Prednosti asinhronosti su mnoge, a između ostalog uključuju mogućnost obrade više zahtjeva istovremeno te podržavanje dugih HTTP konekcija (uključujući HTTP, HTTP / 2 i WebSocket). Djangova asinhronost je između ostalog omogućena i preko Channels knjižnice (https://channels.readthedocs.io/)

9. Uključuje vlastiti ORM maper

Django framework sadržava vlastiti ORM zahvaljujući kojem nije uopće potrebno pisati SQL upita za komunikaciju s bazom podataka kao ni za međusobno definiranje odnosa između različitih objekata.

Tablica 8. Primjer jednostavnog definiranja odnosa jedan-naprama-jedan u Djangu koristeći mogućnosti ORM mapera. Ovdje je definiran odnos kupca i proizvoda. U potpunosti je izbjegnuto pisanje SQL upita

```
from django.db import models


class Kupac(models.Model):
    name = models.CharField(max_length=255)
class Proizvod(models.Model):
    ime = models.CharField(max_length=255)
    kupac = models.OneToOneField(
        Kupac,
        on_delete=models.CASCADE,
        related_name='proizvod'
```

Tablica 9. Primjer postavljanja channel backenda[39] preko Redis baze[40]. Channel backend omogućava povezivanje više servera u jedan komunikacijski sloj.

```
CHANNEL_LAYERS = {
    "default": {
        "BACKEND": "channels_redis.core.RedisChannelLayer",
        "CONFIG": {
            "hosts": [("redis-server-name", 6379)],
        },
    },
```

[39]backend - tj stražni kraj, u web programiranju označava dio programa koji nije dostupan krajnjem korisniku. Između ostalog poslovnu logiku, baze podataka itd.

[40]Redis - Redis je spremište podataka spremljena u memoriji računala otvorenog koda. Može se koristiti kao baza podataka, posrednih poruka i međuspremnik

Tablica 10. Primjer ASGI mogućnosti Djanga, definiranje chat mogućnosti

```
from django.conf.urls import url

from channels.routing import ProtocolTypeRouter, URLRouter
from channels.auth import AuthMiddlewareStack

from chat.consumers import AdminChatConsumer, PublicChatConsumer
from aprs_news.consumers import APRSNewsConsumer

application = ProtocolTypeRouter({

   # WebSocket podržavanje chat-a
   "websocket": AuthMiddlewareStack(
      URLRouter([
         url(r"^chat/admin/$", AdminChatConsumer),
         url(r"^chat/$", PublicChatConsumer),
      ])
   ),

   # Korištenje projekta frequensgi koji pruža APRS protokol
   "aprs": APRSNewsConsumer,

})
```

Tablica 11. Primjer definiranja URL ruta u Djangu. Ovdje je definirana ruta za korisnika i administratora

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
   path('admin/', admin.site.urls),
   path(“,include(user.urls)),
]
```

Tornado

Tornado je Pythonov framework i asinhrona biblioteka za umrežavanje, izvorno razvijen u FriendFeedu (https://www.tornadoweb.org/). Tornado se može prilagoditi na više desetaka

tisuća otvorenih veza, što ga čini idealnim za dugotrajne HTTP veze, WebSockets i druge aplikacije koje zahtijevaju dugotrajnu vezu sa svakim korisnikom.

Tornado se otprilike može podijeliti u četiri glavne komponente:

1. Web okvir

Web okvir uključuje upravljanje zahtjevima te se koristi za izradu web aplikacija

2. Implementacije HTTP-a na strani klijenta i poslužitelja
3. Asinkrona mrežna biblioteka

Služi za komunikaciju preko HTTP protola, te se također može koristit za implementaciju drugih protokola

4. Korutinska biblioteka

Ona omogućava pisanje asinkronog koda na jednostavniji način od lančanih povratnih poziva. To je slično značajci izvorne korutine uvedenoj u Pythonu 3.5 (async def).

Tornado web framework i HTTP poslužitelj zajedno nude potpunu alternativu WSGI-u.

Uzevši u obzir sve navedene značajke nabrojanih Python frameworka prednost pri odabiru se daje onom frameworku koji je brz, pouzdan, siguran, dobro dokumentiran te da podržava ASGI specifikaciju.
Mada je Flask u svojoj izvedbenosti konkurentna i Djangu i Tornadu, obzirom da ne podržava ASGI nije moguće implementirati asinhrone zahtjeve te WebSocket protol.
WebSocket je napredna tehnologija koja omogućuje otvaranje dvosmjerne interaktivne komunikacije između klijenta i poslužitelja. Dakle, pomoću ovog protokola moguće je primati odgovore od servera, bez da se šalje upit serveru. (https://tools.ietf.org)
Primjena WebSocket-a u konstrukciji LIMS-a je mnogobrojna, na primjer, ukoliko se u laboratoriju vrši izvođenje procedura pomoću automatiziranih robota, moguće je dobiti obavijest o završenom postupku u istom trenutku kada je ona i izvršena.

Uzevši u obzir navedeno, kao odabir frameworka se preporučuje Django, obzirom da ima sve mogućnosti kao i Tornado te brojne druge, kao što je vlastiti sustav komunikacije s bazom podataka s čime se u potpunosti uklanja potreba pisanja kompliciranih SQL upita. Nadalje, Django ima iznimno bogatu dokumentaciju, vrlo se lako nadograđuje, ima ugrađene značajke sigurnosti, podržava MVC dizajn te pruža brojne alate preko kojih je razvoj web aplikacija iznimno olakšan i pojednostavljen.

4.3. Odabir tehničke izvedbe komunikacije servera s klijentom

Nakon što smo odabrali arhitekturu izvedbe LIMS-a potrebno je odabrati tehničku izvedbu kojom će se vršiti komunikacija između servera i klijenta. Aplikacijska programska sučelja (API) izlažu svoje usluge ili podatke putem skupa unaprijed definiranih resursa, kao što su metode, objekti ili URI (Stylos i sur., 2009).

Kako je i navedeno u poglavlju uvoda, najčešće korištena aplikacijska programska sučelja su:

- RESTful API
- SOAP
- GraphQL

Uspoređujući SOAP s GraphQL te REST specifikacijom, odmah se vide nedostaci.
Parafrazirajući uvod, SOAP (skraćenica za Simple Object Access Protocol) je specifikacija protokola za razmjenu poruka koji se koristi u implementaciji web usluga.
SOAP je stari protokol koji se oslanjaja na XML za propusne web usluge. Za komunikaciju, SOAP preporučuje WDSL (Web Services Description Language), koji je razvijen od World Wide Web konzorcija. (Soni i Ranga, 2019.)

Sama starost XML-a te WDSL-a utječe na to da se SOAP specifikacija može pronaći generalno samo u starim računalnim sustavima.
Razvijanje daleko lakše čitljivijeg JSON-a (JavaScript Object Notation-a) učinila je XML zastarjelim.

S druge strane, REST specifikacija se može smatrati de facto industrijskim standardnom, no GraphQL s druge strane ipak valja detaljnije usporediti sa REST-om ukoliko predstavlja potencijalno novi standard.

Usporedba REST-a i GraphQL-a

Usporedba REST-a i GraphQL-a nije trivijalna, ponajviše zbog toga što su definirana na različitim razinama apstrakcije. Dok se GraphQL može smatrati tehnologijom, REST podrazumijeva arhitektonski stil kodiranja aplikacija.

Stoga, da bi usporedili GraphQL s REST-om, jedan od načina je GraphQL definirati u arhitektonskom obliku kodiranja programa, poput REST-a.

Pronađene su sljedeće specifikacije pri definiranju GraphQL-a kao arhitektonskog stila kodiranja aplikacija:

1. Klijent/Server komunikacija
2. Komunikacija bez čuvanja stanja
3. Deklarativnost jezika

Najveća razlika između REST-a i GraphQL-a je u deklarativnosti jezika te obliku podataka koji se šalju klijentu. Dok je u REST-u potrebno slati više zahtijeva da bi se došlo do ugniježđenih podataka, kod korištenja GraphQL-a to se može učiniti samo jednim upitom.

Doduše, konačna odluka je ta da je REST poželjniji.
Mada je GraphQL jednako, ako ne i moćniji kao metoda komunikacija servera s klijentom, činjenica da je REST kao metodologija daleko duže u programerskom svijetu je dovoljna da prevlada kao izbor.
Ponajviše što se prilikom izrade LIMS-a moraju svesti potencijalne poteškoće na minimum, obzirom da je sam sistem dovoljno kompleksan, a greške u izradi mogu biti jako skupe tokom vremena.

4.4. Odabir dizajna LIMS-a

Kako je navedeno u uvodu dizajn aplikacije najčešće spada u iduće tri kategorije :
1.MVC - Model View Controller
2. MVP - Model View Presenter
3. MVVM - Model View View Model

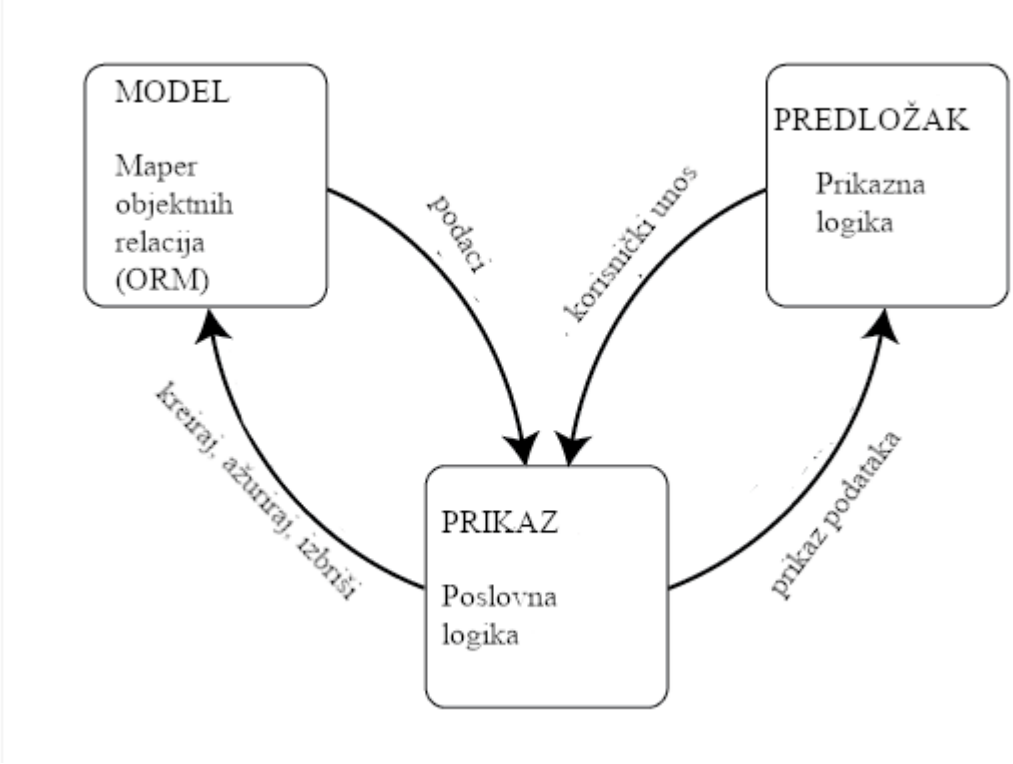
U suštini su sva tri dizajna jako slična, sastoje se od pohrane podataka, upravljača poslovne logike te prikaza aplikacije. Sva tri elementa se nazivaju drugačijim imenima, no svrha im je ista.
Jedina se razlika sastoji u definiranju protoka podataka i toka komunikacije između ta tri dijela aplikacije.

Stoga, se može smatrati da je odabir dizajn-a LIMS-a uglavnom ostavljen na preferencijama programera, mada se ipak prednost može dati MVC (Model/View/Controller) dizajnu obzirom da je najstariji od sva tri.

No, odabir ostaje na MVC (Model/View/Controller) dizajnu ponajviše stoga što je odabirom web-omogućene arhitekture LIMS-a, Python-a kao programskog jezika, Djanga kao framework-a te REST-a kao tehničke specifikacije komunikacije između servera i klijenta odabir MVC dizajna u potpunosti logičan.

Django koristi MVT(Model/View/Template) koji se može smatrati doduše MVC aplikacijskim dizajnom. Razlika je samo u nazivima, pošto je Template sloj u Django-u zapravo View, a View je u suštini Controller.

Podjela MVT (Model/Prikaz/Predložak) dizajna u Django-u je dakle (https://docs.djangoproject.com/):



Slika 23. Pregled MVT dizajna (Model-View-Templete, tj Model, Predložak, Prikaz)

Sloj Modela

Django pruža sloj apstrakcije za strukturiranje i manipuliranje podacima web aplikacije.

Sloj Prikaza

Django ima koncept Prikaza gdje kapsulira logiku odgovornu za obradu korisnikovog zahtjeva i vraćanje odgovora (ima funkciju upravljača u MVC-u).

Sloj Predloška

Sloj predloška pruža dizajnersku sintaksu za pružanje informacija koje treba predstaviti korisniku (ima funkciju Prikaza u MVC-u).

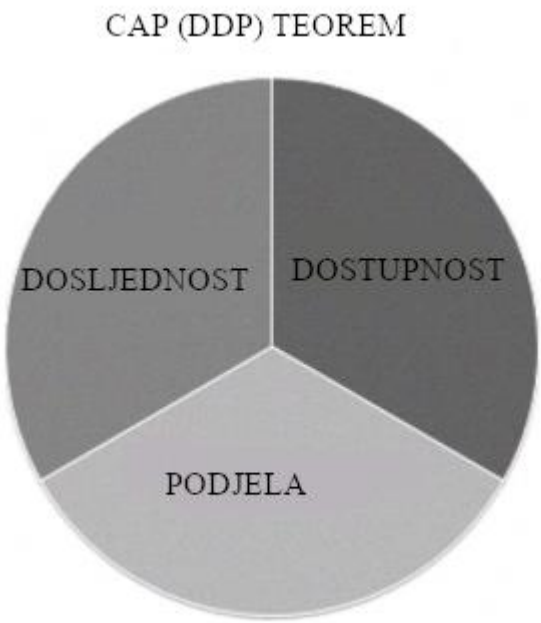
4.5. Odabir baze podataka za LIMS

Prvo pitanje koje se nameće pri odabiru baze podataka je odabir između:

- Relacijskih baza podataka (SQL)
- Nerelacijskih ili NoSQL baza podataka

Relacijske baze podataka su strukturirane kao skup međusobno povezanih tablica, uređene po definiranoj schemi. Nerelacijske baze s druge strane, kao što i ime sugerira ne sadrže nikakvu schemu niti definiranu relaciju između vrijednosti, a najčešće se sastoje od jednostavnog skupa ključ-podatak.

Prije nego što se usporedimo baza podataka, potrebno se upoznati sa osnovnim karakteristikama baze podataka. Te karakteristike daje CAP (Consistency-Aplication-Partition, tj. Dosljednost-Dostupnost-Podjela) teorem koji objašnjava dosljednost, dostupnost i podjelu. (Fekete, 2017)



Slika 24. CAP (DDP) teorem (prilagođeno prema Fekete, 2017)

- Dosljednost:

Znači kad je operacija ažuriranja izvršena, svi mogu pročitati najnoviju verziju podataka iz baza podataka i takav sustav je dosljedan sustav.

- Dostupnost:

Postiže se ako sustav uvijek omogućuje kontinuirani rad. Dostupnost se postiže primjenom baze podataka kao klastera čvorova. U tom slučaju, ako se jedan čvor sruši, drugi čvorovi mogu i dalje raditi.

- Podjela:

Sustav baze podataka koji može raditi čak i ako je jedan od čvorova nedostupan. To se događa preusmjeravanjem svih upita od neakivnog čvora nekom drugom aktivnom čvoru tog sustava

Relacijske baze podataka stavljaju veliku važnost na ispunjavanje navedenog teorema, te podržavaju ACID (Atomicity Consistency Isolation Durability, tj. Atomičnost Dosljednost Izolacija Izdržljivost) svojstva (Rautmare i Bhalerao, 2016):

- Atomičnost

Sve promjene podataka izvode se kao da su jedna operacija. Odnosno, sve promjene se provode, ili nijedna.

Na primjer, u aplikaciji koja prenosi sredstva s jednog računa na drugi, svojstvo atomičnosti osigurava da, ako se uspješno zaduživanje s jednog računa obavi, odgovarajući kredit se daje na drugi račun.

- Dosljednost

Podaci su u konzistentnom stanju kada transakcija započne i kada se izvrši.
Na primjer, u aplikaciji koja prenosi sredstva s jednog računa na drugi, svojstvo dosljednosti osigurava da je ukupna vrijednost sredstava na oba računa ista na početku i na kraju transakcije.

- Izolacija

Prijelazno stanje transakcije je nevidljivo za ostale transakcije. Kao rezultat, čini se da se transakcije izvode serijski.
Na primjer, u aplikaciji koja prenosi sredstva s jednog računa na drugi, izolacija osigurava da neka druga transakcija vidi prenesena sredstva na ili na jednom ili na drugom računu, ali ne u oba, niti u niti jednom.

- Izdržljivost

Nakon uspješne transakcije promjene podataka bivaju zapisane i ne poništavaju se, čak i u slučaju kvara sustava.
Na primjer, u aplikaciji koja prenosi sredstva s jednog računa na drugi, svojstvo trajnosti osigurava da se promjene na svakom računu ne ponište.

S druge strane, NoSQL baze podataka više se usredotočuju na dostupnosti i podjelu te pružaju eventualnu dosljednost.
Oni slijede BASE (Basic Availability Soft state Eventual consistency, tj. Osnovna dostupnost Meko stanje Eventualna dosljednost) svojstva (Rautmare i Bhalerao, 2016):

- Osnovna dostupnost:

NoSQL baza podataka su usredotočene na dostupnost podataka prema zahtjevima CAP teorema

- Meko stanje:

Stanje sustava baze podataka je dinamično i može se mijenjati s vremenom zbog eventualne dosljednosti. Sve kopije baze podataka ne moraju uvijek biti dosljedne

- Eventualna dosljednost:

Nakon bilo koje operacije pisanja, ažuriranja ili brisanja, sustav možda ne odražava odmah provedene izmjene. Ali, s vremenom će postati dosljedno prikazivati izmijenjene podatke u svim kopijama baze.

Uzevši u obzir navedeno NoSQL baze podataka se doimaju manje primamljivije obzir da ne pružaju dosljednost, a transakcije ne podliježu ACID principima.

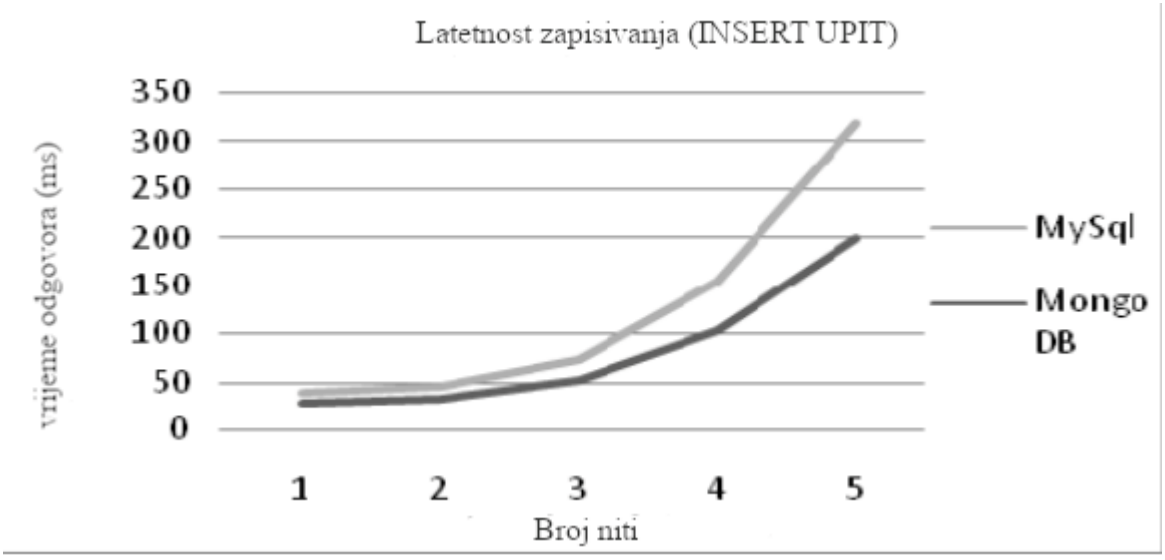
Odabrati ispravnu bazu podataka je iznimno bitno, a glavne razlike gledano iz perspektive web aplikacija su (Rautmare i Bhalerao, 2016):

- Skalabilnost:

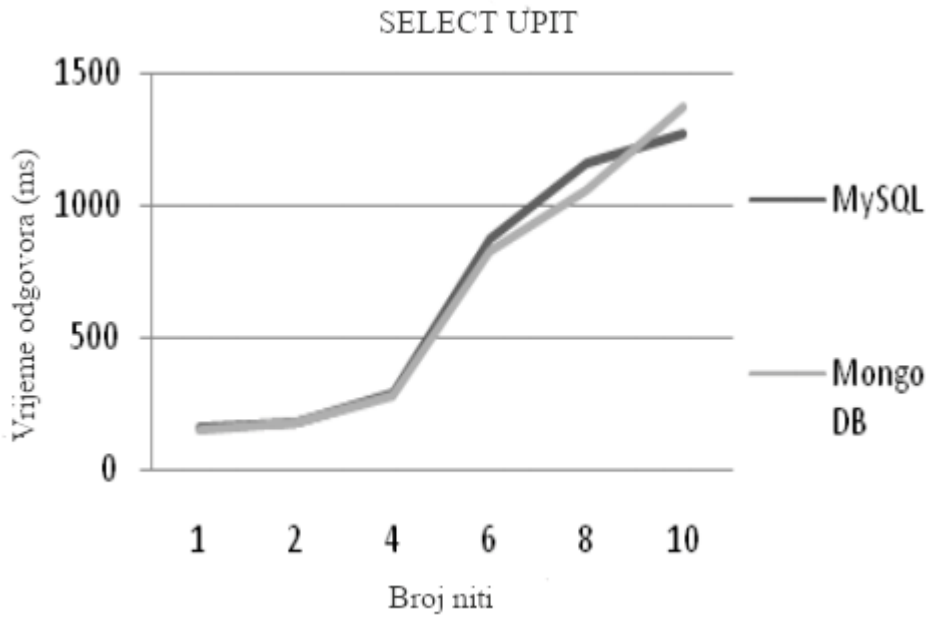
SQL baza podataka podržava okomitu skalabilnost dok NoSQL podržava horizontalnu skalabilnost. Vertikalna skalabilnost odnosi se na sposobnost povećanja izvedbe jednog čvora dodavanjem resursa kao što su memorija ili procesori na isti čvor. U vodoravnoj skalabilnosti, broj čvorova (poslužitelja) se povećava tako da podijeli opterećenje sustava.

- Dohvaćanje podataka:

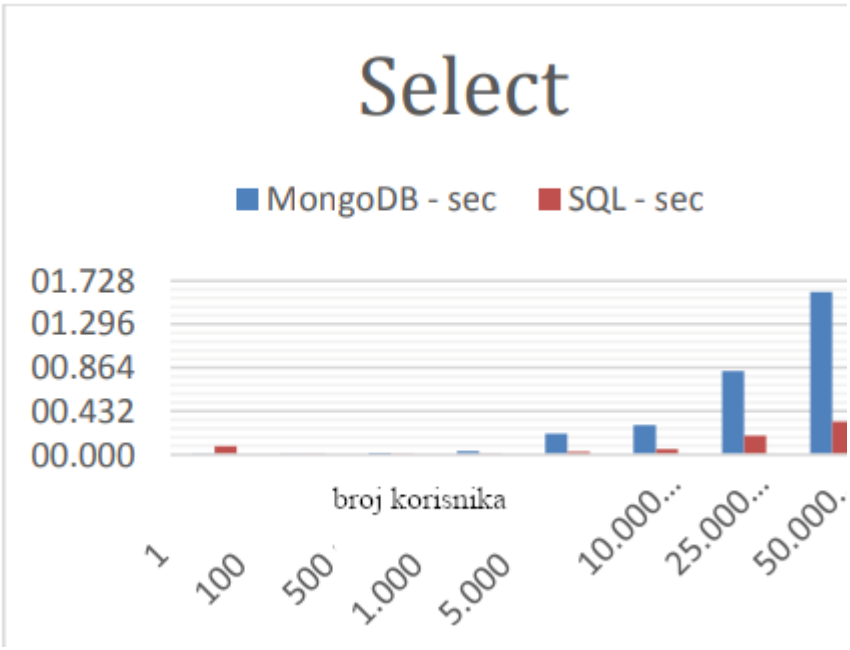
Brzina dohvaćanja podataka je jako bitna kada korisnici trebaju podatke za daljnju obradu. U SQL-u su povezane razne tablice zajedno. Da bi pronašao podatke iz različitih tablica, korisnik mora koristiti JOIN izjave što stvara preglede. Ovo je proces koji zahtijeva dugo vremena. S druge strane, u NoSQL-u su podaci pohranjeni u obliku objekata koji sadržavaju sve povezane podatke. Time se eliminira proces kombiniranja, a zatim prikaz podataka, čime se štedi vrijeme odziva.



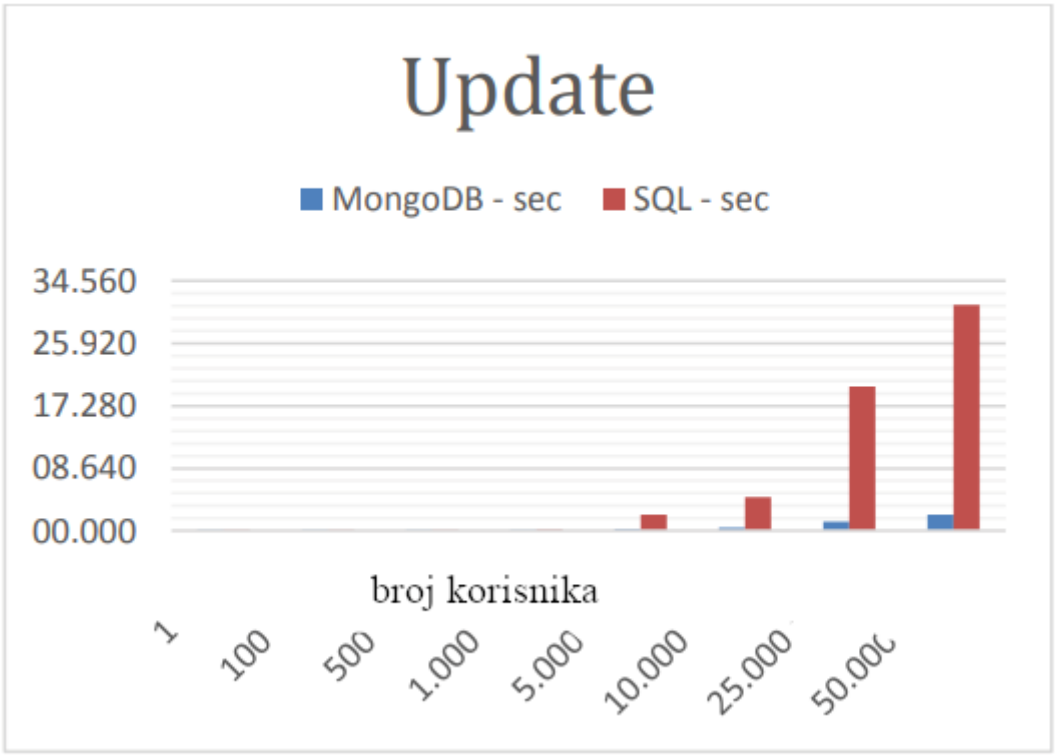
Slika 25. Latentnost zapisivanja ovisno o broju niti, usporedba MySQL baze kao primjera relacijske baze podataka te MongoDB baze, kao primjera NoSQL baze podataka. Vidljiva je veća brzina NoSQL-a (prilagođeno prema Rautmare i Bhalerao, 2016)



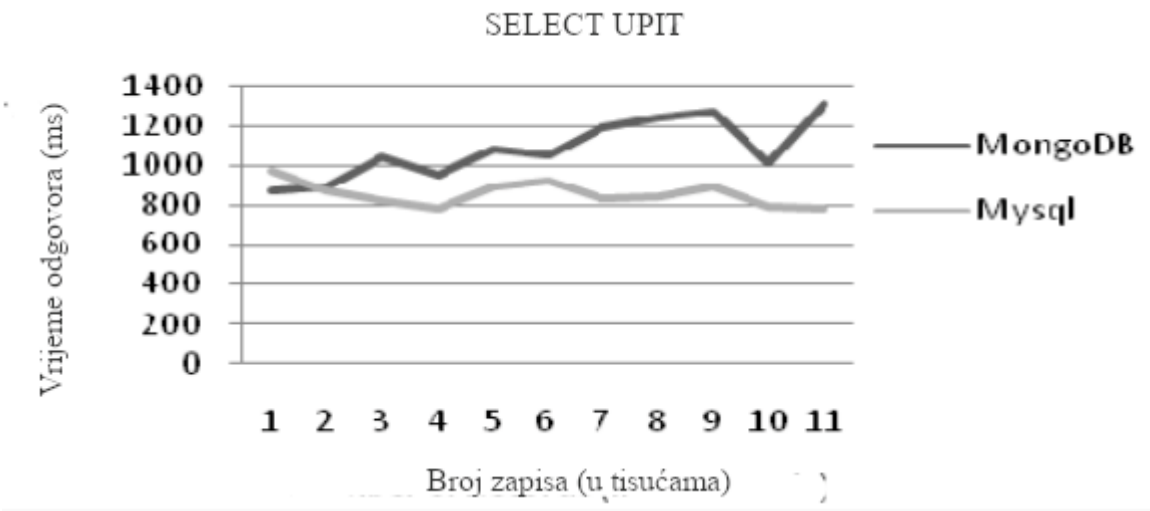
Slika 26. Primjer brzine odgovora ovisno o broju niti, usporedba MySQL baze kao primjera relacijske baze te MongoDB-a kao primjera NoSQL baze podataka. Vidljivo je da je brzina gotovo identična. (prilagođeno prema Rautmare i Bhalerao, 2016)



Slika 27. Brzina select upita ovisno o broju korisnika. usporedba MongoDB-a te SQL-a. Vidljivo je da MongoDB brže obrađuje select zahtjeve prilikom većeg broja istovremenih korisnika (prilagođeno prema Győrödi i sur., 2015)

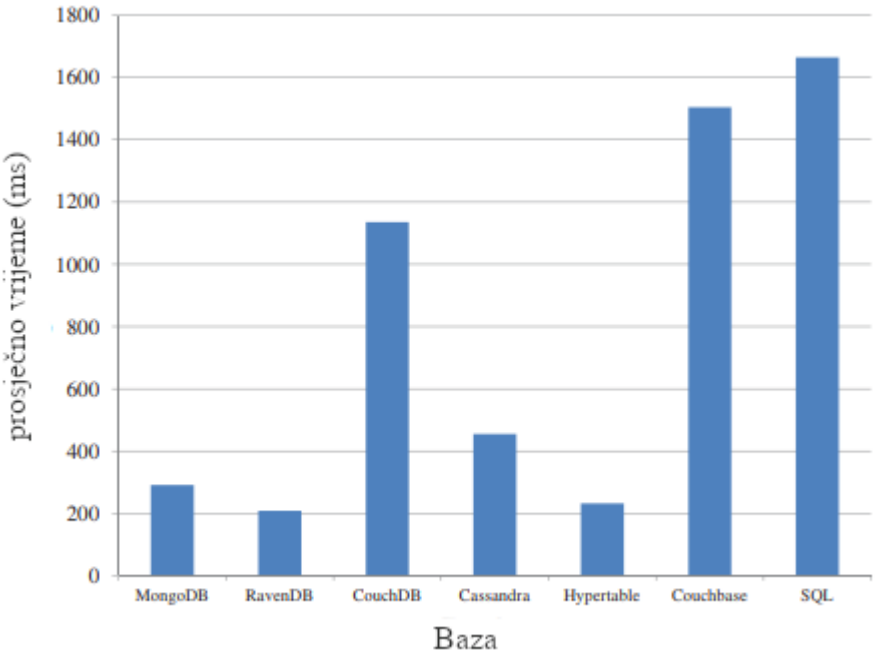


Slika 28. Brzina update upita ovisno o broju korisnika. Usporedba MongoDB-a i SQL-a. Vidljivo je da MongoDB brže obrađuje update upite ovisno o broju istovremenih korisnika (prilagođeno prema Győrödi i sur., 2015)



Slika 29. Vrijeme odgovora, tj. vrijeme obrade i prikaza select upita ovisno o broju zapisa. Usporedba MongoDB-a i MySQL-a. Vidljivo je da s većim porastom broja zapisa

performanse MongoDB-a zaostaju za MySQL-om (prilagođeno prema Rautmare i Bhalerao, 2016)



Slika 30. Prosječno vrijeme instanciranja različitih NoSQL baza podataka usporedno sa SQL bazom podataka. Vidljivo je da su NoSQL baze brže pri instanciranju, tj. pokretanju. (prilagođeno prema Li i Manoharan, 2013)

Tip baze	broj ključeva za dohvatiti					
	10	50	100	1000	10000	100000
MongoDB	4	4	5	19	98	702
RavenDB	101	113	115	116	136	591
CouchDB	67	196	19	173	1063	9512
Cassandra	47	50	55	76	237	709
Hypertable	3	3	3	5	25	159
MS SQL Express	4	4	4	4	11	76

Vrijeme dohvata svih ključeva (ms)

Slika 31. Vrijeme dohvata ključeva različitih NoSQL baza podataka te MS SQL Expressa kao primjera relacijske baze podataka. Vidljivo je da je MS SQL Express znatno brži. (prilagođeno prema Li i Manoharan, 2013)

Tip baze	Broj operacija					
	10	50	100	1000	10000	100000
MongoDB	61	75	84	387	2693	23354
RavenDB	570	898	1213	6939	71343	740450
CouchDB	90	374	616	6211	67216	932038
Cassandra	117	160	212	1200	9801	88197
Hypertable	55	90	184	1035	10938	114872
Couchbase	60	76	63	142	936	8492
MS SQL Express	30	94	129	1790	15588	216479

Vrijeme zapisivanja (ms)

Slika 32. Vrijeme zapisivanja ovisno o broju operacija, pregled različitih NoSQL baza podataka te MS SQL Expressa kao primjera relacijske baze podataka. Vidljivo je da pri porastu broja operacija performanse MS SQL Expressa značajno opadaju, no pri manjem broju operacija (do 100) su iznimno brze (prilagođeno prema Li i Manoharan, 2013)

- Zrelost sustava:

SQL je iskusna tehnologija i stoga su sigurnosne značajke poput provjere autentičnosti, povjerljivosti podataka i integriteta uključeni u SQL. S druge strane, takve sigurnosne značajke se tek trebaju implementirati u NoSQL-u. NoSQL generiraju više problema ili narušavanja sigurnosti zbog nedostatka zrelosti sustava.

Tablica 12. Primjer povezivanja podataka u MongoDB bazi podataka koristeći JavaScript kao programerski jezik. Mada NoSQL baze ne sadrže relacije, njih je potrebno definirati u kodu. Ovo je jednostavan primjer povezivanja datoteke korisnici, kao i polja u datoteci koje ne želimo. Doduše, prilikom složenijih upita, povezivanje zapisa u kodu može postati jako kompliciranо.

```
userSchema.pre(/^find/, function (next) {
  this.populate({
    path: 'korisnici',
    select: '-__v -passwordChangedAt'
  });

  // populate će dohvatiti korisnike i prikazati ih u upitu
  //path je ime datoteke kojeg dohvaćamo, a s - znakom označavamo koja polja ne želimo
  next();
});
```

Uzevši u obzir sve navedene značajke te uspoređujući SQL i NoSQL baze podataka, zaključak je da prednost svakako imaju SQL baze.
Mada su NoSQL baze podataka brže i jednostavnije za skaliranje, nedostatak sigurnosti te njihova nezrelost kao baza podataka ih čine neprihvatljivima za upotrebu u LIMS-u.
Nadalje, mada su NoSQL baze podataka nerelacijske, ipak sadržaju relacije definirane unutar samog koda aplikacije, te kod dovoljno velike web aplikacije zapravo mogu biti kompleksnije od scheme SQL baze podataka. Također, framework Django ne pruža direktno podršku NoSQL bazama podataka te bi se komunikacija morala vršiti preko posredničkog softvera što čini još više potencijalnih sigurnosnih poteškoća. U poslovnom sustavu kod kojeg je dosljednost podataka od iznimne važnosti, SQL baze podataka su jedina opcija.

Potrebno je stoga odabrati među 4 najpopularnije sustava za upravljanje relacijskim bazama podataka:

- Microsoftov SQL Server,
- MySQL,
- Oracle i
- PostgreSQL.

1. Microsoftov SQL Server

Microsoft SQL Server popularan je RDBMS koji se izvodi samo u sustavu Windows platformi. Njegove značajke uključuju jednostavnost uporabe, nisku cijenu i velike performanse.

2. MySQL

MySQL je popularni open source (sustav otvorenog koda) sustav za upravljanje bazama podataka (DBMS) koji je poznat po svojim odličnim performansama. Radi na brojnim operativnim uređajima uključujući većinu Linux inačica. Za poboljšanje performansi, ima vitkiji skup značajki nego kod mnogih drugih DBMS[41]-ova. Njegovi kritičari ističu da nije u potpunosti relacijski DBMS jer ne podržava mnoge ključne značajke relacijske baze podataka, posebno u načinu na koji obrađuje transakcije.

[41]DBMS - Database Management System ili sustav upravljanja bazama podataka

3. Oracle

Oracle je vodeći RDBMS u komercijalnom sektoru. Radi na mnoštvu operativnih sustava i hardverskih platformi. Njegova skalabilna i pouzdana arhitektura učinile su je platformom izbora za mnoge korisnike. Zbog njihove vrlo prilagodljive naravi, Oracle RDBMS-ovi zahtijevaju dobro obučenog administratora baze podataka.

4. PostgreSQL

PostgreSQL je jedan od RDBMS-a s najbogatijim značajki te je otvorenog koda. Postgres je vrlo proširiv. Podržava niz naprednih tipova podataka koji nisu dostupni na MySQL-u (geometrijski / GIS, mrežne adrese, JSONB koji se može indeksirati, nativni UUID, vremenske zone). Ako to nije dovoljno, može dodati i vlastite tipove podataka, operatore i vrste indeksa. Osim bogatog niza značajki, PostgreSQL radi na širokoj razini mnoštva operativnih sustava i hardverskih platformi. (Kline i sur., 2009)

U usporedbi između ovih 4 sustava za upravljanje relacijskim bazama podataka, Microsoftov SQL Server automatski limitira odabir servera, što nije prihvatljivo, te je zatvorenog koda. Oracle, doduše ne ograničava odabir operacijskog sustava, no njegova zloglasna kompleksnost te zatvorenost koda ga ne čine primamljivim. Odluka stoga pada na odabir između MySQL-a te PostgreSQL-a.

Oba sustava su vrlo kvalitetna, no PostgreSQL ima određene prednosti nad MySQL-om koji ga čine primamljivijim, a oni su((https://www.postgresql.org):

- PostgreSQL je objektno-relacijska baza podataka, dok je MySQL čisto relacijska baza podataka
- Bolje podnosi istodobno korištenje baze
- Striktnije se drži SQL standarda
- Sadrži više tipova podataka

Uzevši u obzir sve navedeno, kao baza izbora prevladava PostgreSQL.

5. ZAKLJUČAK

Prednosti korištenja Laboratory Information Management Systema (LIMS-a) u biotehnološkim tvrtkama su toliko brojne da je zaključak da bi se svakako trebali koristiti ukoliko to budžet same tvrtke omogućava.

Prednost se daje samostalnim rješenjima, pošto je svaka biotehnološka tvrtka specifična te ima svoje već uhodane radne obrasce koje bi bilo teško mijenjati za potrebe komercijalnog rješenja, također, izvedba LIMS-a podešena potrebi svake tvrtke osigurava da će sustav na kraju i biti korišten, što je jedna od najbitnijih stavki.

Nadalje, kako se uzima u obzir sve veća međusobna povezanost laboratorija, web-omogućen LIMS se nameće kao logično rješenje. Također, web-omogućen LIMS je jeftiniji, pošto zahtijeva minimalno komercijalnog software-a za izvedbu te se dijeli na tri dijela: poslužitelja, baze podataka te klijentova.

Sva poslovna logika se nalazi na jednom poslužitelju, na kojem se nalazi i aplikacija za upravljanje zahtjevima korisnika, dok je baza podataka također odvojena.

Međusobna odvojenost poslovne logike, podataka te krajnjih korisnika omogućava da se troškovi održavanja smanje iznajmljujući komercijalne poslužiteljske farme kao komercijalne spremišne prostore. Mada se sigurnost može izdvojiti kao primjedba ovog rješenja, obzirom da su svi dijelovi aplikacije međusobno odvojeni, čak i da sigurnost pojedine komponente bude ugrožena, ne bi bilo moguće steći potpunu sliku same izvedbe te procesa koji su u nj uključeni.

Prednost web-omogućenog LIMS-a je ta što klijent može biti običan web-poslužitelj, nije potrebno izrađivati kompleksna i često nespretna sučelja. Ta mogućnost potencira i da samo korisničko sučelje bude platformski-agnostično, što znači da bi osobe mogle imati pristup LIMS-u na bilo kojem obliku računala, od osobnih računala do pametnih telefona, u suštini na bilo kojem uređaju koji može pokrenuti web-preglednik.

Nadalje, jedna od najbitnijih stavki pri izradi LIMS-a je odabir programerskog jezika. Python se pronašao kao jezik izbora, već je korišten uvelike u znanstvenom svijetu, iznimno je jednostavan te je brzina pisanja koda vrlo velika. Također, analizom literature našlo se brojna postojeća izvedbena rješenja koja koriste Python što je samo potvrdilo njega kao odabir.

Iduća stavka je bila odabir frameworka-a (programskog okvira) u kojem će se poslužiteljski dio aplikacije napisati.

Pri analizi mogućih framework-a Django se pokazao najboljim, od broja ljudi koji ga već koriste, njegovog objektno-relacijskog mapera koji olakšava komunikaciju s bazom podataka, od detaljnosti dokumentacije kao i činjenice da podržava ASGI specifikaciju što ga čini asinhronim,

Asinhronost Djanga omogućava korištenje WebSocket protokola kao i dugih HTTP veza, konkretna primjena je dobivanje informacija o izvedenog laboratorijskom procesu u realnom vremenu, kao i mogućnost povezivanja raznih laboratorijskih uređaja koji bi mogli vršiti dvostranu komunikaciju sa poslužiteljem.

Obzirom da Django podržava Model/View/Controller (Model/Prikaz/Upravljač) dizajn, time se taj dizajn i prihvatio kao najboljim.

Za komunikaciju poslužiteljske aplikacije sa klijentom odabran je REST. REST metodologija olakšava samo kodiranje, povećava preglednost same aplikacije programerima te pruža jednostavno sučelje klijentima za konzumiranje podataka. REST je također i industrijski standard pri izradi aplikacijskih programerskih sučelja.

Velika se važnost također pridodala i odabiru baze podataka, pošto je to jedna od najbitnijih stavki u LIMS-u.

Mada su relacijske baze podataka bile bez konkurencije godinama došlo je do pojave nerelacijskih ili NoSQL baza. Navedene prednosti NoSQL baza podataka su nepostojanje scheme, što ih čini fleksibilnijima kao i njihova brzina.

No, NoSQL ne podliježu jednim od najbitnih stavki kvalitetne baze podataka, a to je dosljednost podataka. Naime, ukoliko korisnik izbriše određeni podatak, neki drugi korisnik koji koristi bazu podataka u isto vrijeme ga može vidjeti, jer promjene u bazi nisu atomične. Nadalje, mada nepostojanje scheme, tj. međusobnog povezivanja podataka čini na prvi pogled NoSQL baze fleksibilnijima, ipak je potrebno stvoriti relacije, no u kodu. Ta činjenica dovodi do toga da kompleksnost logike kod dovoljno velike aplikacije postane jako velika i čak nepregledna.

Zaključak je da su relacijske baze podataka već testirane tokom desetljeća, a njihova sigurnost, dosljednost i industrijska prihvaćenost ih čine jedinim mogućim odabirom za korištenje u LIMS-u. Također, Django nudi jednostavnost komunikacije sa relacijskim bazama podataka pomoću svog objektno-relacijskog mapera, gdje se komunikacija vrši preko upita napisanih u Pythonu, a ne preko SQL jezika što je mnogo jednostavnije. Od najčešće korištenih sustava za upravljanje relacijskim bazama podataka odabran je PostgreSQL zbog stvor striktnijeg držanja SQL standarda te veće broja nativnih oblika podataka, čime se olakšava spremanje podataka u bazu.

6. LITERATURA

Analysis of a laboratory information management system (LIMS) ,Dan Bentley, 1999., http://www.umsl.edu/~sauterv/analysis/LIMS_example.html, pristupljeno 23.3.2020.

Data structures, 2020., https://docs.python.org/3/tutorial/datastructures.html, pristupljeno 23.3.2020.

ASGI (Asynchronous Server Gateway Interface) specification, 2020., https://asgi.readthedocs.io/en/latest/specs/main.html#abstract, pristupljeno 17.2. 2020.

ASGI documentation, 2020., https://asgi.readthedocs.io/en/latest/ , pristupljeno 17.2. 2020.

About postresql, 2020., https://www.postgresql.org/about/ , pristupljeno 10.2.2020.

Anderson C. The model-view-viewmodel (MVVM) design pattern. U: Pro business applications with silverlight 5. Sebastopol, O'Reilly Media, 2012, str. 461-499.

Applications for Python, 2020., https://www.python.org/about/apps/ , pristupljeno 10.2. 2020.

Becht H, Staples J. Soft typing of general first-order languages. APSEC '95, Brisbane, 1995, 480-488.

Chodorow K, Dirolf M. MongoDB: The definitive guide. Sebastopol, O'Reilly Media, 2015, str. 5-23.

Hello, ASGI, Christie T, 2018., https://www.encode.io/articles/hello-asgi , pristupljeno 27.3.2020.

Craig T, Holland R, D'Amore R, Johnson JR, Mccue HV, West A, Caddick M. Leaf LIMS: A flexible laboratory information management system with a synthetic biology focus. *ACS Synth Bio,* 2017, 6, 2273-2280.

Django, 2020., https://www.djangoproject.com/, pristupljeno 17.3.2020.

Documentation, 2020., https://docs.djangoproject.com/en/3.0/ , pristupljeno 10.2.2020.

Extensions, 2020., https://asgi.readthedocs.io/en/latest/extensions.html#http-2-server-push, pristupljeno 17.3.2020.

Fekete A . CAP Theorem. U: Encyclopedia of database systems, Liu L, Özsu MT, New York, Springer-Verlag, 2017, 1-1.

Fielding RT. Architectural styles and the design of network-based software architectures. 2000, str. 76-105.

Fransen SM, Nyrup A. Implementation of a ‘standard’ LIMS. *Laboratory Automation & Information Management*, 1998, 33, 227-233.

Gabor B, Kemme B. Exp-WF:Workflow support for laboratory information systems. ICDEW'06, Atlanta, 2006, 1-9.

Gibbon GA. A brief history of LIMS. *Laboratory Automation & Information Managemen*t, 1996, 32, 1-5.

Gibbons J. Structured programming in Java. *ACM SIGPLAN Notices*, 1998, 33, 40-43.

GraphQL., 2020., https://github.com/graphql/ , pristupljeno 10.2.2020.

Graphql - Graphql/graphql-spec. 2020., https://github.com/graphql/graphql-spec , pristupljeno 20.3.2020.

Gyorödi C, Gyorödi R, Sotoc R. A comparative study of relational and non-relational database models in a web- based application. *IJACSA*, 2015, 6, 78-83.

Han JEH, Le G, Du J. Survey on NoSQL database. ICPCA 2011, Port Elizabeth, 2011, 363-366.

History of PHP – manual, 2020., https://www.php.net/manual/en/history.php.php , pristupljeno 20.3.2020.

Holzner S. Beginning ruby on rails. Birmingham, Wrox, 2007, str. 33-63.

Hughes-Croucher T, Wilson M. Node: up and running. Sebastopol, O'Reilly Media, 2012, str. 33-55.

Java™ programming language, https://docs.oracle.com/javase/8/docs/technotes/guides/language/index.html , pristupljeno 17.3.2020.

Kline KE, Kline D, Hunt B. SQL in a nutshell. Sebastopol, O'Reilly Media, 2009, str. 16-59.

Kolva D, Paszko C, Post J. Considerations in selecting a laboratory information management system for the small laboratory. *Chemom Intell Lab Syst*, 1994, 26, 171-174.

Leff A, Rayfield J. Web-application development using the model/view/controller design pattern. IEEE EDOC'01, Seattle, 2001, 118-127.

Lei K, Ma Y, Tan Z. Performance comparison and evaluation of web development technologies in PHP, Python, and Node.js. CSE2014, Chengdu, 2014, 661-668.

Li Y, Manoharan S. A performance comparison of SQL and NoSQL databases. PACRIM, Singapore, 2013, 15-19.

Massé M. REST API design rulebook. Sebastopol, O'Reilly Media, 2012, str. 1-11.

Mcdowall R, The role of laboratory information management systems (LIMS) in analytical method validation. *Anal Chim Acta*, 1999, 391, 149-158.

Melo A, Faria-Campos A, Delaat D, Keller R, Abreu V, Campos S. SIGLa: An adaptable LIMS for multiple laboratories. *BMC Genomics*, 2010, 11(Suppl 5).

Node.js – About, 2020., https://nodejs.org/en/about/ , pristupljeno 17.3.2020

NumPy, 2020., https://numpy.org/ , pristupljeno 17.3.2020.

Oluwatosin HS. Client-server model. *IOSR-JCE*, 2014, 16, 57-71.

PEP 3333 -- Python web server gateway interface v1.0.1. , 2020., https://www.python.org/dev/peps/pep-3333/ , pristupljeno 17.3.2020.

Page- Python useful models, 2020., https://wiki.python.org/moin/UsefulModules, 23.3.2020.

Php, 2020., https://github.com/php/php-src, pristupljeno 23.3.2020.

Piggee C. LIMS and the art of MS proteomics. *Anal Chem*, 2008, 80, 4801-4806.

Prasad PJ, Bodhe G. Trends in laboratory information management system. *Chemom Intell Lab Syst,* 2012, 118, 187-192.

The Python SQL toolkit and object relational mapper, 2020., https://www.sqlalchemy.org/ , pristupljeno 17.3.2020.

The Python standard library, 2020., https://docs.python.org/3/library/ , pristupljeno 27.3.2020

Rautmare S, Bhalerao DM. MySQL and NoSQL database comparison for IoT application. ICACA, Coimbatore, 2016, 235-238.

Robillard M, Walker R, Zimmermann T. Recommendation systems for software engineering. *IEEE Software*, 2010, 27, 80-86.

Ruby, 2018., https://www.ruby-lang.org/en/about/ , pristupljeno 27.3.2020.

Ruby on rails, 2017., https://rubyonrails.org/ , pristupljeno 24.3. 2020.

Sarmenta LFG. Bayanihan: Web-based volunteer computing using Java. WWCA'98, Tsukuba, 1998, 444-461.

SciPy library, 2020., https://scipy.org/scipylib/index.html , pristupljeno 17.2.2020.

Severance C, Roy T. Fielding. Understanding the REST style. *Computer*, 2015, 48, 7-9.

Sheehan RJ. Teaching operating systems with ruby, *ACM SIGCSE Bulletin*, 2007, 39, 38.

Soni A, Ranga V. API features individualizing of web services: REST and SOAP. *IJITEE*, 2019, 8, 664-671.

Steele TW, Laugier A, Falco F. The impact of LIMS design and functionality on laboratory quality achievements. *Accreditation Qual Assur,* 1999, 4, 102-106.

Stylos J, Faulring A, Yang Z, Myers BA. Improving API documentation using API usage information. VL/HCC 2009, Corvallis, 2009, 119-126.

Tornado, 2020., https://www.tornadoweb.org/en/stable/ , pristupljeno 17.2.2020.

Troshin PV, Postis VL, Ashworth D, Baldwin SA, Mcpherson MJ, Barton GJ. PIMS sequencing extension: A laboratory information management system for DNA sequencing facilities. *BMC Res Notes*, 2011, 4.

Ulma W, Schlabach DM. Technical considerations in remote LIMS access via the world wide web. *J Automat Meth Manag Chem,* 2005, 4, 217-222.

Visualization with Python, 2020., https://matplotlib.org/ , pristupljeno 24.3.2020.

The websocket protocol, 2020., https://tools.ietf.org/html/rfc6455 , pristupljeno 17.2.2020.

Welcome to ecma international, 2017., http://www.ecma-international.org/ , pristupljeno 23.3.2020.

Welcome to Flask, 2020., https://flask.palletsprojects.com/en/1.1.x/ , pristupljeno 24.3.2020.

Welcome to Python.org., 2020., https://www.python.org/about/ , pristupljeno 24.3.2020.

What can PHP do?, 2020., https://www.php.net/manual/en/intro-whatcando.php , pristupljeno 17.2.2020.

What is JavaScript?, 2020., https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript , pristupljeno 17.2.2020.

Zadka M. Installing Python. U: DevOps in Python. New York, Apress, 2019, str. 1-6.

Zhang Y, Luo Y. An architecture and implement model for model-view-presenter pattern. ICCSIT 2010, Chengdu, 2010, 532-536.

7. SAŽETAK/SUMMARY

U okviru ovog rada istražene su prednosti, korisnost te sama svrhovitost Laboratory Information Management Systema (LIMS-a) za potrebe biotehnološki tvrtki.
Nakon detaljnog pregleda svih aspekata i parametara koje treba uzeti u obzir pri izradi LIMS a došlo se do zaključka da bi najbolje rješenje bio web-omogućen LIMS, koji bi se podijelio na tri dijela: poslužitelja, bazu podataka te klijenta.
Za poslužiteljski dio aplikacije odabran je programski jeziku Python, a koristio bi se Django frameworkom te MVC dizajnom.
Za upravljanje i čuvanje podataka se odabrao PostegreSQL koji je relacijski sustav za upravljanje bazama podataka.

Within this framework of this thesis, usefulness and purposefulness of using Laboratoy Information Management System (LIMS) have been researched.
After the detailed overlook of all aspects and parameters that need to be taken into consideration before building the LIMS , the conclusion is that the best solution would be a web-based one, which would be divided into three parts: server, database and clients.
For the server side od the application, Python has been chosen as a preferred programming language as well as Django web framework which utilizes MVC design.
As a preferred database used for storage and managing the data, PostegreSQL has been found to be the best choice, and is also a relational database management system.

Sveučilište u Zagrebu
Farmaceutsko-biokemijski fakultet
Studij: Farmacija
Zavod za biokemiju i molekularnu biologiju
A. Kovačića 1, 10000 Zagreb, Hrvatska

Diplomski rad

VAŽNOST, SVRHOVITOST TE PRIMJER TEHNIČKE IZVEDBE LABORATORY INFORMATION MANAGEMENT SYSTEMA U BIOTEHNOLOŠKIM TVRTKAMA

Domagoj Bakota

SAŽETAK

U okviru ovog rada istražene su prednosti, korisnost te sama svrhovitost Laboratory Information Management Systema (LIMS-a) za potrebe biotehnološki tvrtki.

Nakon detaljnog pregleda svih aspekata i parametara koje treba uzeti u obzir pri izradi LIMS a došlo se do zaključka da bi najbolje rješenje bio web-omogućen LIMS, koji bi se podijelio na tri dijela: poslužitelja, bazu podataka te klijenta.

Za poslužiteljski dio aplikacije odabran je programski jeziku Python, a koristio bi se Django frameworkom te MVC dizajnom.

Za upravljanje i čuvanje podataka se odabrao PostegreSQL koji je relacijski sustav za upravljanje bazama podataka.

Rad je pohranjen u Središnjoj knjižnici Sveučilišta u Zagrebu Farmaceutsko-biokemijskog fakulteta.

Rad sadrži:	83 stranica, 32 grafičkih prikaza, 12 tablica i 69 literaturnih navoda. Izvornik je na hrvatskom jeziku.
Ključne riječi:	LIMS, server/klijent, Python, baze podataka, REST
Mentor:	**Dr. sc. Gordan Lauc**, *Redoviti profesor Sveučilišta u Zagrebu Farmaceutsko-biokemijskog fakulteta.*
Ocjenjivači:	**Dr. sc. Gordan Lauc**, *Redoviti profesor Sveučilišta u Zagrebu Farmaceutsko-biokemijskog fakulteta.* **Dr. sc. Toma Keser**, *viši asistent Sveučilišta u Zagrebu Farmaceutsko-biokemijskog fakulteta* **Dr. sc. Jasna Jablan**, *docent Sveučilišta u Zagrebu Farmaceutsko-biokemijskog fakulteta*

Rad prihvaćen: svibanj 2020.

University of Zagreb
Faculty of Pharmacy and Biochemistry
Study: Pharmacy
Department of biochemistry and molecular biology
A. Kovačića 1, 10000 Zagreb, Croatia

Diploma thesis

IMPORTANCE, PURPOSEFULNESS AND EXAMPLE OF TECHNICAL IMPLEMENTATION OF LABORATORY INFORMATION MANAGEMENT SYSTEM FOR BIOTECHNICAL COMPANIES

Domagoj Bakota

SUMMARY

Within this framework of this thesis, usefulness and purposefulness of using Laboratoy Information Management System (LIMS) have been researched.

After the detailed overlook of all aspects and parameters that need to be taken into consideration before building the LIMS , the conclusion is that the best solution would be a web-based one, which would be divided into three parts: server, database and clients.

For the server side od the application, Python has been chosen as a preferred programming language as well as Django web framework which utilizes MVC design.

As a preferred database used for storage and managing the data, PostegreSQL has been found to be the best choice, and is also a relational database management system.

The thesis is deposited in the Central Library of the University of Zagreb Faculty of Pharmacy and Biochemistry.

Thesis includes: 83 pages, 32 figures, 12 tables and 69 references. Original is in Croatian language.

Keywords: LIMS, server/client, Python, database, REST

Mentor: **Gordan Lauc**, **Ph.D.** *Full Professor*, University of Zagreb Faculty of Pharmacy and Biochemistry

Reviewers: **Gordan Lauc**, **Ph.D.** *Full Professor*, University of Zagreb Faculty of Pharmacy and Biochemistry

Toma Keser, **Ph.D.** *Associate Professor*, University of Zagreb Faculty of Pharmacy and Biochemistry

Jasna Jablan, **Ph.D.** *Associate Professor*, University of Zagreb Faculty of Pharmacy and Biochemistry

The thesis was accepted: May 2020.