

УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ



Милош Арсић

РАД СА БАЗАМА ПОДАТАКА У
ПРОГРАМСКОМ ЈЕЗИКУ ПАЈТОН СА
ПРИМЕНАМА У БИОИНФОРМАТИЦИ

мастер рад

Београд, 2025.

Ментор:

др Весна МАРИНКОВИЋ, ванредни професор
Универзитет у Београду, Математички факултет

Чланови комисије:

др Јована КОВАЧЕВИЋ, ванредни професор
Универзитет у Београду, Математички факултет

др Нина РАДОЛИЧИЋ МАТИЋ, доцент
Универзитет у Београду, Математички факултет

Датум одбране: _____

*Захваљујем се својој менторки, професорки
Весни Маринковић, која је уз велико стрпљење и
конструктивне савете пратила израду овог рада.*

*Велику захвалност дугујем и професорки
Јовани Ковачевић, која ми је помогла својим
сугестијама при формирању идеје рада.*

*Захваљујем се колегиници Јелени Поповић,
професорки биологије, на рецензији рада.*

*Највећу захвалност дугујем својој
породици, пријатељима и колегама,
на свакодневној подршци и мотивацији.*

*Овај рад посвећујем свима који су допринели
мом професионалном и личном развоју.*

Наслов мастер рада: Рад са базама података у програмском језику Пајтон са применама у биоинформатици

Резиме: Апликативни SQL се добија уметањем SQL наредби у програмске језике вишег нивоа. На тај начин се омогућава повезивање програма написаних на програмским језицима вишег нивоа са базом података чиме се проширује скуп могућности за рад са базама података у различитим системима. Како програмски језик Пајтон има широку употребу у области биоинформатике постоји потреба да се биолошки подаци који се чувају у бази података приказују и обрађују већ развијеним алгоритмима коришћењем одговарајућих апликација. У раду је приказано на који начин се могу премостити разлике између релационог модела података и модела података у програмском језику Пајтон. Описане су и коришћене постојеће наредбе и модули програмског језика Пајтон који омогућавају повезивање и рад са системом за управљање базом података MariaDB на локалном серверу. Поред тога развијена је апликација за рад са биолошким подацима који се чувају у бази података. Помоћу програма *Clustal Omega* и *FastTree* реализовано је вишеструко поравнање и конструисано је филогенетско стабло за узорке вируса SARS-CoV-2. Графички кориснички интерфејс апликације је креиран коришћењем библиотеке *Tkinter*.

Кључне речи: апликативни SQL, релационе базе података, биоинформатика, поравнање секвенци, филогенетско стабло

Садржај

1	Увод	1
2	Релационе базе података	2
2.1	Потреба за базама података у програмима	2
2.2	Развој упитног језика	3
2.3	Систем за управљање релационим базама података	4
2.4	Релациони модел података	6
2.5	Шема релационе базе података Библиотека	11
3	Програмирање базе података система MariaDB у програм- ском језику Пајтон	13
3.1	Подешавање локалног сервера	14
3.2	Повезивање са базом података	16
3.3	Прављење курсора и табела	17
3.4	Унос, брисање и ажурирање података	19
3.5	Упити над подацима у бази података	22
4	Преглед биоинформатичких појмова	26
4.1	Обрада биолошких података	26
4.2	Поравнавање нуклеотидних секвенци	29
4.3	Филогенетска анализа	33
5	Развој апликације за рад са биолошким подацима	37
5.1	Вирус SARS-CoV-2	38
5.2	Структура базе података и прављење табела	39
5.3	Преузимање, припрема и унос биолошких података у базу података	41
5.4	Графички кориснички интерфејс	43
5.5	Преглед функционалности апликације	46

САДРЖАЈ

5.6	Обрада грешака	56
5.7	Приказ и анализа добијених резултата	57
6	Закључак	59
	Литература	60

Глава 1

Увод

Уметањем SQL наредби у програме писане на неком од језика вишег нивоа омогућено је повезивање програма са базом података. Тако настаје апликативни SQL (енг. *embedded SQL*) чијом се употребом проширује скуп могућности за руковање базом података, али и за рад програма којима је за чување неопходних информација потребна база података [1]. Поред значајног доприноса у виду рачунских могућности, помоћу апликативног SQL-а може се повећати безбедност, а и избећи логичке грешке у раду са базом података [2].

Програмски језик Пајтон представља програмски језик опште намене. Припада језицима вишег нивоа са интуитивном и једноставном синтаксом. Иако је користан за аутоматизацију једноставних послова, Пајтон нуди могућности и за решавање сложенијих проблема [3], због чега је постао најчешће коришћен програмски језик у свету данас [4].

Циљ овог рада је да илуструје на који начин се коришћењем наредби програмског језика Пајтон може повезивати и радити са базом података. Поред тога, у раду ће бити показано како се коришћењем библиотеке *Tkinter* може направити графички кориснички интерфејс (енг. *graphical user interface*) помоћу кога се могу приказивати резултати рада, а и побољшати интеракција корисника са програмима писаним на програмском језику Пајтон. На крају ће, у последњем сегменту овог рада, бити приказан проблем поравнања секвенци као и филогенетске анализе у биоинформатици. На практичном примеру биће показано на који начин је могуће развити апликацију написану у програмском језику Пајтон која ће радити на поравнавању дугих нуклеотидних секвенци сачуваних у бази података. Додатно, на примеру апликације биће илустрован рад са базом података у којој се чувају биолошки подаци.

Глава 2

Релационе базе података

2.1 Потреба за базама података у програмима

Израда система за рад са подацима поред писања програма обухватала је, све до појаве првих база података, и израду помоћних датотека - скупова слогова коришћених за чување података. У периоду од 1884. до 1951. године у раду са подацима су коришћене бушене картице, а са појавом првих електричних рачунара за чување података коришћене су магнетне траке и од 1955. године магнетни дискови. Нови медијуми омогућили су знатно сложенију обраду података, али су поставили и нове захтеве [5].

Велике организације су, за потребе аутоматизације послова, развијале концепт аутоматског управљачког система (енг. *automated file management system*) који се састојао од више помоћних датотека за чување података и више програма који су те податке обрађивали. Код оваквих система било какво проширење односило се на писање додатних програма и израду нових пратећих датотека за чување података. Ипак, овакав концепт је у пракси показао велики број недостатака међу којима су и повећање редуцатности услед великог броја понављања података у различитим датотекама, неконзистентност услед измена или брисања, зависност од програма за обраду и од начина организације података на систему, неразвијен опоравак у случају изненадног прекида рада система као и слабу ефикасност приликом истовременог коришћења од стране више од једног корисника [5].

2.2 Развој упитног језика

Отклањање поменутих недостатака вршено је кроз више различитих фаза. Процес припрема за прво слетање летелице Аполо на Месец почео је 1960. године и обухватао је обраду велике количине података за које до тада није постојао адекватан систем за чување и обраду. Тим поводом је авијација Северне Америке (енг. *North American Aviation*) основала метод кратко назван GUAM (енг. *Generalized Update Access Method*). IBM се прикључује Авијацији Северне Америке како би се овај метод развио у систем за управљање информацијама (енг. *Information Management System*) [1].

Током шездесетих година двадесетог века развија се интегрисано чување података (енг. *Integrated Data Store*) засновано на мрежном моделу. У његовом развоју учествовао је Чарлс Бахман, а за приказ сложених односа у бази коришћене су хијерархијске структуре. Тако 1967. године настаје радна група која је учествовала у стандардизацији база података те се истичу три дисјунктна језика - језик за дефинисање података (енг. *Data Definition Language*), језик за дефинисање подшема (енг. *Subschema Data Definition Language*) и језик за манипулисање подацима (енг. *Data Manipulation Language*). Први језик је омогућио администраторима базе података да дефинишу шему, други је омогућио апликацијама да дефинишу делове базе док је трећи је омогућио рад са подацима у бази. Овај приступ је имао доста зависности међу подацима, није био теоријски заснован и имао је сложен приступ подацима [6].

Како би се поменути недостаци превазишли 1970. године Едгар Франк Код у својим радовима предлаже *концепт релационог модела података* као најједноставније структуре опште намене, која има највећи могући степен независности података [7]. Како би дефинисао релациони модел Код је развио два упитна језика који су били засновани на логици првог реда, а који су омогућили приступ подацима [8].

Под утицајем поменутих радова у истраживачкој лабораторији Сан Хозе 1970. године настаје пројекат *Систем R*. Циљ пројекта био је да се покаже да је релациони модел могуће имплементирати [6].

Проучавајући релациони модел Доналд Чемберлин и Рејмонд Бојс развијају упитни језик SEQUEL (енг. *Structured English Query Language*). SEQUEL представља упитни језик дизајниран тако да ради са подацима који су сачувани у оригиналном Систему R који је развио IBM [7].

Чемберлин и Бојс су веровали да је могуће направити упитни језик који би био приступачан и људима који немају формално математичко образовање, али и језик који је могуће једноставно уносити путем тастатуре. Овај језик био је декларативан - у упитима се описује информација која се тражи за разлику од процедуралних језика код којих се описује начин доласка до решења [8].

Након тестирања SEQUEL се даље развија као део пројекта Систем R, али се, због проблема са заштитним знаком, 1977. његов назив скраћује у SQL (енг. *Structured Query Language*). Током 1970-тих и 1980-тих развијају се и прве комерцијалне имплементације међу којима су и DB2 и Oracle [8].

Амерички национални институт за стандарде је 1986. године дефинисао SQL стандард под називом SQL-86 стандард. Настанак SQL стандарда допринео је развоју и широкој употреби релационих база података. Стандардом је обезбеђен језик за управљање и манипулацију подацима у бази података што је омогућило премошћавање разлика и пренос података између различитих произвођача база података. Осим тога, стандард је допринео развоју и одржавању база података, повећао је њихову поузданост и смањио могућност за настајање грешака [8].

SQL стандард је такође омогућио развој широког спектра алата и апликација за управљање и манипулацију базама података, укључујући управљање трансакцијама, оптимизацију упита, анализу података и друго. Данас се SQL користи у широком спектру апликација, укључујући веб апликације, пословне апликације, софтвер за управљање инвентаром, апликације за анализу података и друге. SQL је доживео неколико ажурирања и побољшања кроз различите верзије стандарда, укључујући SQL-86, SQL-89, SQL-92, SQL:1999, SQL:2003, SQL:2006, SQL:2008, SQL:2011, SQL:2016, SQL:2019 и SQL:2023.

2.3 Систем за управљање релационим базама података

Један од првих радова који су допринели развоју система за управљање релационим базама података објавио је 1970. године Едгар Франк Код. У њему је представљен релациони модел података, његове предности и примена у раду са великим базама података [9]. Касније се, као што је наведено у поглављу 2.2, као покушај да се имплементира релациони модел, развија Систем R чијим се даљим унапређивањем развија SQL као стандардни језик за управљање релационим базама података [1, 8].

Прве комерцијалне имплементације система за управљање релационим базама података настају крајем седамдесетих и током осамдесетих година двадесетог века. То су биле база података компаније Оракл (енг. *Oracle Database*), IBM DB2 и Sybase. Ове имплементације допринеле су популаризацији релационих база података и њиховој широј употреби [10].

Технике објектно-оријентисаног програмирања почињу да се користе током осамдесетих година двадесетог века и у раду са базама података. Тако настају и развијају се *системи за управљање објектно-оријентисаном базом података* (енг. *Object-Oriented Database Management System*). Поред тога што су имале боље перформансе, ове базе података су биле флексибилније и њихово одржавање је било јефтиније. Концепти наслеђивања и полиморфизам су знатно смањили број релација у бази података. Неке од њих су развијане како би ефикасно могле да се повезују са програмима писаним у неком конкретном програмском језику објектно-оријентисане парадигме, док су друге имале развијен и програмски језик који је служио за рад са базом. Неки од проблема у раду са објектно-оријентисаним базама података били су превазилажење разлике између објектно-оријентисаног и релационог модела и зависност дизајна базе података од дизајна апликације у којој се она користи. Како би се недостаци отклонили почетком деведесетих година двадесетог века развија се *објектно-релациони модел* (енг. *Object Relational Database Model*) и систем за управљање објектно-релационим базама података (енг. *Object Relational Database Management System*). Многи елементи овог модела постају саставни део SQL-а. Примери система за управљање релационим базама података који су засновани на објектно-релационом моделу података су Oracle, IBM DB2, Microsoft SQL Server и PostgreSQL [11].

Интеграција техника објектно-оријентисаног програмирања у релационе базе података има велики значај у развоју база података. Иако су релационе базе података заступљене данас, верује се да ће, са повећањем количине података које је потребно чувати у бази података, објектно-оријентисане базе података наставити да добијају на значају, јер нуде бољу организацију података и већу ефикасност при раду са базама података великих размера [12].

Током 1995. године у Шведској настаје MySQL као систем за управљање базама података отвореног кода. MySQL представља једну од најчешће коришћених база до данас [13]. У августу 2000. године, као библиотека у програмском језику C, настаје SQLite.

Због своје поузданости, једноставности и лаке интеграције, SQLite је врло брзо стекао популарност и постао такође један од широко коришћених система за управљање базом података [14].

Након што је компанија Оракл купила компанију Сан Мајкростистемс (енг. *Sun Microsystems*) у јануару 2009. године, из система за управљање базом података MySQL настаје систем MariaDB. Између ова два система за управљање базама података постоји компатибилност те већина апликација које су развијене за MySQL могу да се користе и са системом за управљање базом MariaDB без икаквих измена. Захваљујући отвореном коду и јавно доступној документацији, MariaDB представља погодан избор за израду истраживачких и академских радова и за развој апликација отвореног кода [15].

2.4 Релациони модел података

Основна структура у релационом моделу је *релација*, али како бисмо формално увели појам релације дефинишемо следеће појмове:

- *Атрибут* - именовано својство (колона табеле); означава се са A_i , $i \in \mathbb{N}$.
- *Домен* - скуп свих дозвољених вредности неког атрибута A_i ; означава се са D_i или са $dom(A_i)$.
- *Шема релације* представља структуру $R(A_1, A_2, \dots, A_N)$, где је R назив релације, а чини је коначан скуп атрибута $\{A_i \mid i = \overline{1, N}\}$, за неко $N \in \mathbb{N}$, и коначан скуп ограничења над скупом вредности које атрибути могу да узму. Другим речима, шема релације представља коначан скуп парова облика *атрибут-домен* односно $R = \{(A_1 : D_1), (A_2 : D_2), \dots, (A_N : D_N)\}$, за неко $N \in \mathbb{N}$. У једној шеми релације називи атрибута морају да буду јединствени. Шема релације мора да садржи бар један атрибут.
- *Степен* - број атрибута шеме релације.
- *Торка* - један елемент Декартовог производа $D_1 \times D_2 \times \dots \times D_N$, где је сваки $D_i = dom(A_i)$, кад $i = \overline{1, N}$ за неко $N \in \mathbb{N}$.
- *Релација* се дефинише над шемом релације. Нека је $R(A_1, A_2, \dots, A_N)$ шема релације и $D_i = dom(A_i)$, $i = \overline{1, N}$. Релација r над шемом релације R је коначан подскуп Декартовог производа $D_1 \times D_2 \times \dots \times D_n$.

Другим речима, релација представља било који коначан скуп торки. Релације су често приказане у виду табела док торке представљају редове у табели. Редослед торки није битан, али не могу да постоје две идентичне торке у истој релацији.

- *Кардиналност релације* - број торки неке релације.
- *Шема релационе базе података* - коначан скуп шема релација $R = \{R_j \mid R_j \text{ је шема релације, } j = \overline{1, M}\}$, за неко $M \in \mathbb{N}$, и коначан скуп ограничења U која важе између њих.
- *Релациона база података* - коначан скуп релација над неком шемом релационе базе података R односно $rbp = \{r_j \mid r_j \text{ је релација, } j = \overline{1, M}\}$, за неко $M \in \mathbb{N}$.

За непознате или непостојеће вредности атрибута дефинише се универзална вредност која се означава са *NULL* [5].

Поред поменутих *структурних ограничења*, релациони модел има и компоненту која се односи на *интегритет релационог модела базе података*, која служи за представљање ограничења која важе у моделу. Сва ограничења можемо да поделимо у три категорије:

- *Интегритет ентитета* - јединственост торки у релацијама захтева постојање правила за задавање вредности атрибута на основу којих би се свака торка разликовала.
- *Ограничења над атрибутима шема релација* - у оквиру шема релација постоје скупови ограничења којима се описују дозвољене вредности атрибута шеме релације.
- *Референцијални интегритет* - у оквиру шеме релационе базе података постоји скуп ограничења који може да укључује и атрибуте из више релација, а на основу кога се дефинишу дозвољене вредности атрибута једне релације у зависности од постојећих вредности атрибута друге релације.

Како бисмо прецизније описали интегритетску компоненту релационог модела дефинишимо још неке појмове. У следећим дефиницијама користимо ознаку R за шему релације $R(A_1, A_2, \dots, A_N)$, $N \in \mathbb{N}$ и ознаку r за релацију дефинисану над шемом релације R .

Дефиниција 2.4.1 (супер-кључ) *Супер-кључ шеме релације R представља било који подскуп скупа атрибута који на једнозначан начин одређује торке релације r .*

Како свака торка у релацији мора да буде јединствена, то свака шема релације има бар један подскуп скупа атрибута који је супер-кључ. Један супер-кључ био би и скуп свих атрибута шеме релације. Међутим, често постоји и мањи подскуп атрибута за који ово важи и нама ће бити од интереса да одредимо што мањи такав подскуп.

Дефиниција 2.4.2 (кандидат-кључ) *Кандидат-кључ шеме релације R представља било који подскуп скупа атрибута који на једнозначан начин одређује торке у релацији r такав да ни један његов прави подскуп не одређује торке релације r на једнозначан начин.*

Приметимо да се одређивањем могућег кандидат-кључа скуп атрибута којим би се на једнозначан начин идентификовале торке релације r смањује и постаје најмањи, у смислу инклузије, подскуп скупа атрибута шеме релације R . Кандидат-кључ не мора да буде јединствен. Од свих кандидат-кључева бирамо један који ће постати главни идентификатор сваке торке.

Дефиниција 2.4.3 (примарни-кључ) *Примарни кључ шеме релације R представља један њен изабрани кандидат-кључ.*

Приликом одабира примарног кључа предност се даје оним кандидат-кључевима који у стварности најприродније идентификују торке посматране релације. Уколико су нека два кандидат-кључа према том критеријуму једнака, предност се даје оном који заузима мање меморије.

Ако су релације добро дефинисане, помоћу вредности атрибута торке једне релације може да се референцира на торке других релација. Како кандидат-кључ на јединствени начин одређује сваку торку у једној релацији, коју зовемо циљна релација, довољно је да се вредности атрибута који су у саставу кандидат-кључа у полазној релацији поклапају са вредностима одговарајућих атрибута друге (реферишуће) релације. Зато се дефинише и појам страниг кључа.

Дефиниција 2.4.4 (страни-кључ) *Страни кључ у шеми релације R је било који подскуп скупа њених атрибута који има вредности неког кандидат-кључа друге релације или универзалну вредност $NULL$.*

Вредност страног кључа у циљној релацији не мора увек да буде позната. Сада могу да се прецизније опишу интегритет ентитета и референцијални интегритет у релационом моделу.

Услов интегритета ентитета подразумева да ни један атрибут шеме релације R који је у саставу примарног кључа не сме имати вредност NULL у релацији r . У строжем облику, услов интегритета ентитета не ограничава се само на примарни кључ, већ укључује све кандидат-кључеве [16].

Референцијални интегритет представља услове који се односе на вредности страног кључа. Атрибути који су у саставу страног кључа морају да имају вредности које одговарају вредностима одговарајућих атрибута који су у саставу примарног кључа циљне релације (релације у којој ти атрибути чине примарни кључ) или вредност NULL, ако је она дозвољена. Поред тога, током ажурирања података у релацијама (током брисања и измене) мора се очувати поменуто својство, али се поред тога у циљним релацијама приликом брисања не сме обрисати торка или изменити део кандидат-кључа који представља страни кључ у некој другој релацији без претходне провере о додатном губитку информација. Зато дефинишемо статичке и динамичке услове референцијалног интегритета [16].

Статички услов референцијалног интегритета подразумева да сваки подскуп скупа атрибута шеме релације R који представља страни кључ може да у релацији r има само вредности примарног кључа из циљне релације или вредност NULL [5].

Динамички услов референцијалног интегритета дефинише се на следећи начин: нека је r релација над шемом релације R и нека је s реферишућа релација над шемом релације S која садржи страни кључ K . За сваку од операција уклањања торки или ажурирања вредности атрибута који су у саставу примарног кључа у релацији r наводи се једна од четири пропратне акције над торкама реферишуће релације код којих страни кључ одговара примарном кључу циљне релације:

- *NO ACTION* - операција се не извршава ако у реферишућој релацији постоје одговарајуће торке, а иначе се операција извршава;
- *CASCADE* - операција се увек извршава, али се из релације s уклањају одговарајуће торке или се ажурира вредност атрибута који су у саставу страног кључа (каскадно брисање или ажурирање);

- *SET NULL* - операција се увек извршава, али се у одговарајућим торкама релације *s* вредност атрибута који су у саставу страног кључа поставља на вредност NULL;
- *SET DEFAULT* - операција се увек извршава, али се у одговарајућим торкама релације *s* вредност атрибута који су у саставу страног кључа поставља на подразумеване вредности дефинисане у шеми релације.

У случају да се за неки страни кључ не наведе динамичка спецификација услова рефернцијалног интегритета подразумевана акција је NO ACTION [16].

Манипулативна компонента релационог модела служи за описивање поступака за одржавање и коришћење података. Поступци одржавања података су описани кроз елементарне операције уноса и уклањања торке из релације, измене вредности једног атрибута торке, али и увид у садржај једне торке у релацији. Поступци коришћења података засновани су на предикатској логици првог реда. Постоје две варијанте за формално изражавање поступака коришћења података у релационом моделу: релациона алгебра и релациони рачун. По узору на манипулативну компоненту релационог модела развили су се и упитни језици за рад са базом података [16].

Како би се редуковао скуп ограничења над атрибутима класичног релационог модела (скаларност и елементарност) крајем седамдесетих година двадесетог века се развија и угњежђени релациони модел (енг. *nested relational model*). Овај појам превазилази оквире овог рада и заинтересовани читалац више може да прочита у књизи [17].

2.5 Шема релационе базе података Библиотека

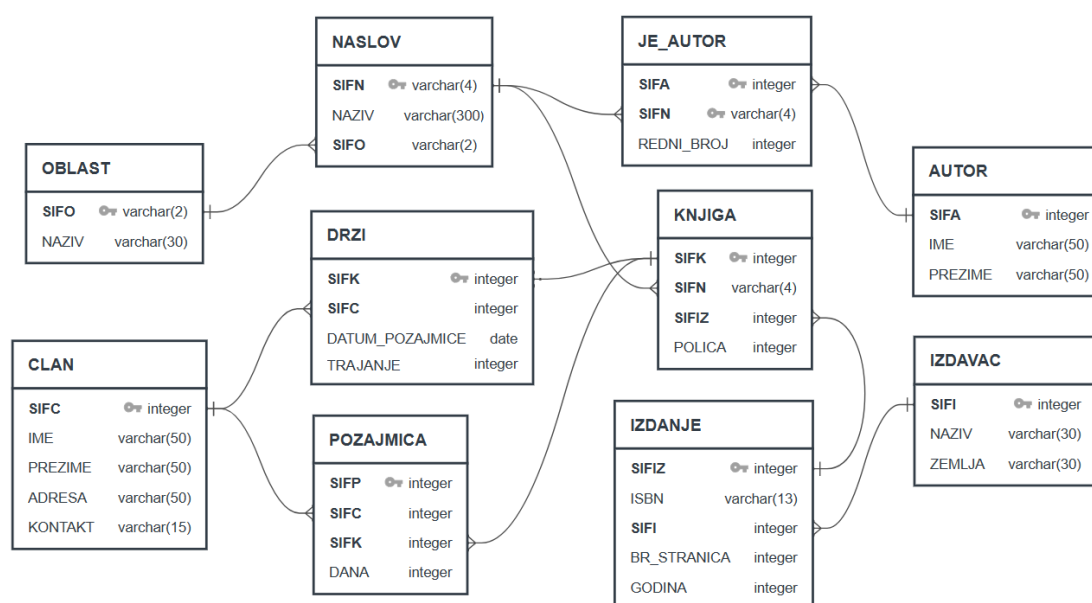
У поглављу 3 овог рада ће у примерима бити коришћена база података Библиотека која је направљена по узору на базу података представљену у [5]. База података Библиотека чува информације о члановима, књигама, областима, насловима, ауторима, издавачима, издањима и позајмицама у библиотеци. Састоји се од десет табела. Табела 2.1 садржи списак свих табела и кратак преглед и опис њихових атрибута.

назив табеле	преглед и опис атрибута
OBLAST	шифра области и назив области
NASLOV	шифра наслова, назив наслова и шифра области којој наслов припада (страни кључ на табелу OBLAST)
AUTOR	шифра аутора и његово име и презиме
JE_AUTOR	шифра аутора (страни кључ на табелу AUTOR), шифра наслова (страни кључ на табелу NASLOV) и редни број који описује који је то аутор по реду, уколико постоји више аутора истог наслова
CLAN	шифра члана библиотеке, подаци о имену, презимену, адреси и контакт телефону члана
IZDAVAC	шифра издавача, назив и земља у којој је издавач регистрован
IZDANJE	целобројна шифра издања, ISBN (енг. <i>International Standard Book Number</i>), шифра издавача (страни кључ на табелу IZDAVAC), број страница и година издања
KNJIGA	шифра књиге, шифра наслова (страни кључ на табелу NASLOV), шифра издања (страни кључ на табелу IZDANJE) и редни број полице на коју је књига распоређена
DRZI	шифра књиге (страни кључ на табелу KNJIGA), шифра члана (страни кључ на табелу CLAN), датум почетка и трајање позајмице
POZAJMICA	шифра позајмице, шифра члана (страни кључ на табелу CLAN), шифра књиге (страни кључ на табелу KNJIGA) и број дана колико је књига била позајмљена

Табела 2.1: Преглед и опис атрибута у бази података Библиотека

Табела `JE_AUTOR` повезује ауторе са насловима њихових дела. У табели `DRZI` чувају се информације о тренутно активним позајмицама, док су у табели `POZAJMICA` информације о реализованим (завршеним) позајмицама.

На слици 2.1 приказана је база података Библиотека са типовима атрибута и означеним релацијама између табела. Примарни кључ је означен малом ознаком кључа поред типа атрибута.



Слика 2.1: Приказ базе Библиотека

У поглављу 3, како би се боље приказало програмирање база података у програмском језику Пајтон, у релационој бази података система MariaDB биће имплементирана база података Библиотека. Такође ће бити приказани разноврсни примери упита и наредби које се могу користити за рад са базом података. Детаљи имплементације базе података Библиотека неће бити приказани у потпуности зато што превазилазе оквире овог рада и биће део његове пратеће документације.

Глава 3

Програмирање базе података система MariaDB у програмском језику Пајтон

Програмски језик Пајтон представља језик објектно-оријентисане парадигме који је настао на институту CWI (*Centrum Wiskunde & Informatica*) у Холандији [3]. Због своје преносивости и једноставне синтаксе програмски језик Пајтон се у пракси показао као добар избор за ефикасно учење техника програмирања и до данас је достигао велику популарност [4].

У биоинформатици, о којој ће бити више речи у поглављу 4, програмски језик Пајтон представља један од најчешће коришћених програмских језика. Пајтон нуди велики број библиотека за научно израчунавање, што олакшава рад са биолошким подацима. Једноставна интеграција са другим алатима и подршка заједнице додатно потврђују значај програмског језика Пајтон у биоинформатичким истраживањима [18].

Током израде рада разматране су различите врсте база података и због своје приступачности изабрана је база података MariaDB, која као што је већ речено у поглављу 2.3, представља систем за управљање релационим базама података отвореног кода настао из система за управљање базом података MySQL.

У овом поглављу биће приказани неки од начина на које се програм написан у програмском језику Пајтон може повезивати са базом података MariaDB.

3.1 Подешавање локалног сервера

Архитектура клијент-сервер представља модел у коме један рачунар, кога ћемо у тексту звати *клијент*, захтева путем мреже ресурсе од другог рачунара, кога ћемо звати *сервер*. Поред тога на мрежи може бити више клијентских рачунара који истовремено шаљу захтеве једном или већем броју серверских рачунара, који раде заједно како би испунили захтеве клијената. Сервер обично има базу података за чување података и покреће програме за обраду захтева.

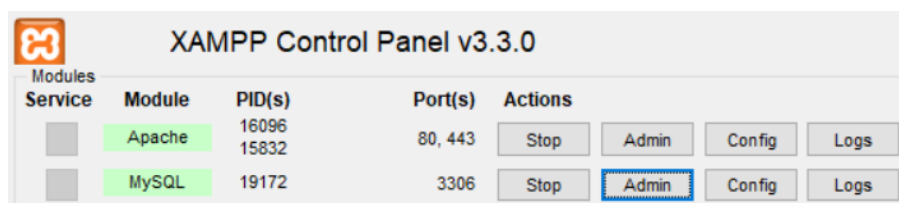
За комуникацију клијента и сервера постоје протоколи као што су *протокол за пренос датотека* (енг. *File Transfer Protocol*), *протокол за једноставну размену поште* (енг. *Simple Mail Transfer Protocol*) и *протокол за пренос хипертекста* (енг. *Hypertext Transfer Protocol*). Ова архитектура омогућава размену података и користи се у многим апликацијама, за е-пошту, код база података и на интернету [19].

За подешавање локалног сервера може се користити софтвер **XAMPP**, бесплатан програм отвореног кода, чији је назив акроним састављен од речи: мултиплатформски (енг. *cross-platform (X)*), Apache (A), MariaDB (M), PHP (P) и Perl (P). Из самог назива се може видети да **XAMPP** користи Apache, који представља познати сервер за хостовање веб сајтова и веб апликација. Једна од његових функционалности је да прихвата захтеве корисничког претраживача (клијента) и шаље одговоре у облику веб страница, слика или других садржаја. Поред тога алат **XAMPP** садржи и друге компоненте неопходне за рад веб сервера па сервер и клијент не морају да буду на различитим рачунарима. Посетиоци серверу приступају путем веб-прегледача, који комуницира са Apache сервером који је покренут помоћу програма **XAMPP** [20].

Након инсталације¹ и покретања потребно је, кликом на дугме *Start* активирати Apache, а потом и систем за управљање базом података (што је MySQL за програм **XAMPP**).

Након успешног покретања, називи Apache и MySQL биће обојени зеленом бојом и могу се видети и информације о покренутим процесима и придруженим портovima. Поред тога доступна су и дугмад за приступ додатним подешавањима. Изглед успешно покренутог сервера и система за управљање базом података приказан је на слици 3.1.

¹Радна верзија програма **XAMPP** која је коришћена у овом раду је 8.0 и може се преузети путем линка [21].



Слика 3.1: Успешно покренут сервер и систем за управљање базом података

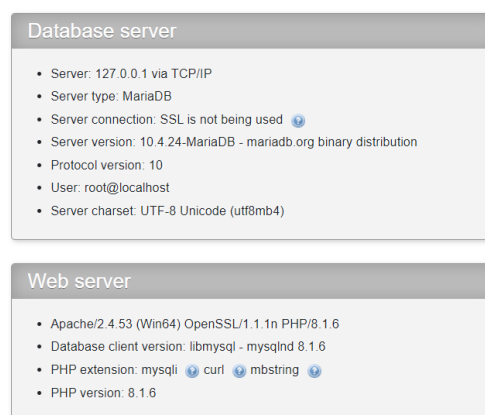
Приступ страници за администрацију система за управљање базом података омогућен је притиском на дугме *Admin* које се налази у линији која одговара сервису MySQL. Алтернативно, страници за администрацију је могуће приступити и путем веб-прегледача, путем адресе:

`http://localhost/phpmyadmin/`

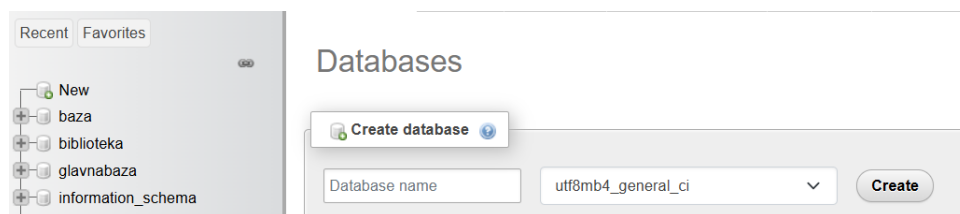
Информације о систему за управљање базом података и о веб серверу видљиве су у десном углу почетне странице. Информације о систему који је коришћен у овом раду приказане су на слици 3.2.

Израда базе на покренутом локалном серверу може да се реализује кликом на дугме **New** у левом углу екрана. Потребно је унети назив базе и одабрати колацију (енг. *collation*) - начин на који се текстуални подаци чувају, сортирају и упоређују у бази података. У нашем случају, за израду базе **Библиотека**, биће одабрана колација: `utf8mb4_general_ci` која омогућава рад са UTF-8 кодирањем знакова и слабијим несензитивним поређењем података. База података се прави притиском на дугме **Create** и након тога биће приказана у списку доступних база података у листи са леве стране екрана као што је илустровано на слици 3.3.

За израду апликације коришћена је верзија Пајтона 3.8 и развојно окружење Python IDLE које може да се преузме путем линка [22].



Слика 3.2: Информације о систему за управљање базом података и о серверу



Слика 3.3: Прављење базе података и приказ доступних база података

3.2 Повезивање са базом података

Повезивање са базом података реализовано је употребом драјвера MySQL Connector, који представља драјвер за повезивање са MySQL серверима. MySQL Connector омогућава програмима написаним на програмском језику Пајтон да приступају MySQL бази података помоћу програмског интерфејса апликације (енг. *Application Programming Interface*) који је у складу са спецификацијом PEP 249 (*Python Database API Specification v2.0*) [23]. Инсталација драјвера путем командне линије као и његово ажурирање реализује се путем две наредбе:

```
$> pip install mysql-connector-python
$> pip install mysql-connector-python --upgrade
```

За повезивање са базом података потребно је направити објекат класе `MySQLConnection` или позвати методу `mysql.connector.connect` [24].

У програму за рад са базом података је направљена функција под називом `povezivanje_sa_bazom` која користи четири неопходна податка за повезивање са базом података која се налази на локалном серверу. Она позива функцију `connect()` како би се направио објекат везе програма са базом података. Пример имплементације те функције приказан је у листингу 3.1.

```
def povezivanje_sa_bazom(hname, uname, upass, dbname):
    bp = mysql.connector.connect(
        host = hname,
        user = uname,
        password = upass,
        database = dbname)

    return bp
```

Листинг 3.1: Функција за повезивање са базом података

Уколико се веза са базом података не оствари, програм се прекида и добија се грешка `InterfaceError`, коју је потребно даље обрадити у главном програму.

У главном програму се позива креирана функција и у променљивој `veza` се чува веза са базом података Библиотека, што је приказано у листингу 3.2.

```
veza = povezivanje_sa_bazom("localhost", "root",  
                             "", "biblioteka")
```

Листинг 3.2: Позив функције за повезивање са базом података

3.3 Прављење курсора и табела

Комуникација са базом података није могућа све док се над објектом везе не направи *објекат курсора*. Курсори представљају посреднички механизам који омогућава рад са базом података - унос података, покретање упита и читање њихових резултата ред по ред или у блоковима [25].

Објекат курсора правимо позивањем методе `cursor()` над објектом везе, што је приказано у листингу 3.3.

```
veza = povezivanje_sa_bazom("localhost", "root",  
                             "", "biblioteka")  
kursor = veza.cursor()
```

Листинг 3.3: Прављење курсора над објектом везе

Прикажимо поступак прављења једне табеле базе података Библиотека. SQL наредба за прављење табеле `CLAN` приказана је у листингу 3.4.

```
CREATE TABLE CLAN (  
    SIFC INT NOT NULL PRIMARY KEY,  
    IME VARCHAR(50) NOT NULL,  
    PREZIME VARCHAR(50) NOT NULL,  
    ADRESA VARCHAR(50) NOT NULL,  
    KONTAKT VARCHAR(15) NOT NULL);
```

Листинг 3.4: SQL наредба за прављење табеле `CLAN`

Најчешће коришћена метода којом се покреће SQL наредба или упит над неким курсором је `execute`. Ова метода има више доступних варијанти које ће бити описане и коришћене у раду. Најједноставније је да се као први параметар методе `execute` проследи ниска која представља SQL наредбу или упит.

Уколико SQL наредбу приказану у листингу 3.4 доделимо као ниску променљивој `sql` потребно је само над курсором позвати методу за извршавање SQL наредби чиме ће табела `CLAN` бити креирана.

Позивање методе `execute` са SQL упитом приказано је у листингу 3.5.

```
kursor.execute(sql)
veza.commit()
kursor.close()
veza.close()
```

Листинг 3.5: Позивање методе `execute`

Потребно је да се остварена веза са базом података, на крају програма или након што нам она није више потребна, затвори што се реализује позивањем методе `close()` над објектом везе.

Пре тога је, уколико је опција аутоматског потврђивања (енг. *auto-commit*) искључена, потребно потврдити све измене над базом података позивањем методе `commit()` над објектом везе.

Приказане наредбе могу да се изврше само једном - након првог извршавања наредбе за прављење табеле она ће постојати у бази података и при покушају да се она поново направи биће пријављена грешка. Како би се обрадила и препознала грешка потребно је укључити модул `errorcode` који се налази у заглављу `mysql.connector`.

Пример начина за обраду грешке приликом прављења табеле приказан је у листингу 3.6.


```
try:
    kursor.execute(sql)
except mysql.connector.Error as e:
    if e.errno == errorcode.ER_TABLE_EXISTS_ERROR:
        print("Neuspeh! Tabela postoji.")
    else:
        print(e.msg)
else:
    print("Tabela je napravljena.")
veza.commit()
kursor.close()
veza.close()
```

Листинг 3.6: Обрада грешке

Поступак брисања табеле или ажурирања њене структуре сличан је приказаном. Потребно је одговарајућу SQL наредбу проследити као ниску методи `execute`.

У пројектној документацији овог рада налази се програм `biblioteka.py` за креирање целокупне базе Библиотека на локалном серверу.

3.4 Унос, брисање и ажурирање података

Унос, брисање и ажурирање података представљају важне операције за рад са базом података. Слично поступку прављења, измене структуре и брисању табела, све SQL наредбе за манипулацију са подацима могу да се употребљавају и у наредбама у програмском језику Пајтон. Уколико су подаци за унос познати пре извршавања програма онда је један од начина за њихов унос у табелу сличан описаном поступку за прављење табеле - потребно је проследити SQL наредбу као ниску методи `execute`. Уколико подаци нису познати пре извршавања програма (учитавају се са улаза или из датотеке), може се направити припремљена SQL наредба за унос којој се приликом позивања методе `execute` може придружити торка са одговарајућим подацима.

Пример таквог уноса приказан је у листингу 3.7.

```
veza = povezivanje_sa_bazom("localhost", "root",
                             "", "biblioteka")
kursor = veza.cursor()
sql = (
    "INSERT INTO clan (SIFC,IME,PREZIME,ADRESA,KONTAKT)"
    "VALUES (%s, %s, %s, %s, %s)"
ulaz = input().split(",")
podaci = (2307, ulaz[0].strip(), ulaz[1].strip(),
          ulaz[2].strip(), ulaz[3].strip())
try:
    kursor.execute(sql, podaci)
except mysql.connector.Error as e:
    print(e.msg)
else:
    print("Unos je uspesan!")
veza.commit()
kursor.close()
veza.close()
```

Листинг 3.7: Унос података у табелу CLAN

Приказани програмски сегмент може да се изврши само једном. Уколико након првог успешног извршавања поново покренемо програм добијемо грешку `Duplicate entry '2307' for key 'PRIMARY'` због понављања вредности примарног кључа. У следећем потпоглављу рада биће приказан један од начина за решавање овог проблема.

Уколико желимо да истовремено извршимо једну SQL наредбу за више различитих података може се позвати метода `executemany` којој се као први параметар прослеђује једна припремљена SQL наредба, а потом као други параметар листа торки за које је потребно извршити прослеђену SQL наредбу.

Метода `executemany` позива методу `execute` за сваку торку из прослеђене листе података. У листингу 3.8 приказане су наредбе којима се читају подаци о члановима из датотеке `podaci.txt` и извршава се њихов унос у табелу `CLAN` позивањем методе `executemany`.

```
veza = povezivanje_sa_bazom("localhost", "root",
                             "", "biblioteka")

kursor = veza.cursor()
sql = (
    "INSERT INTO clan (SIFC,IME,PREZIME,ADRESA,KONTAKT)"
    "VALUES (%s, %s, %s, %s, %s)"
)
podaci = []
i = 2308
with open('podaci.txt', 'r', encoding='utf-8') as fajl:
    for linija in fajl:
        linija = str(i)+","+linija
        print(linija)
        elementi = [l.strip()
                     for l in linija.strip().split(',')]]
        if len(elementi) == 5:
            i+=1
            podaci.append(tuple(elementi))
try:
    kursor.executemany(sql, podaci)
except mysql.connector.Error as e:
    print(e.msg)
veza.commit()
kursor.close()
veza.close()
```

Листинг 3.8: Позивање методе executemany

У пројектној документацији рада налази се програм `podaci_biblioteka.py` чијим се извршавањем најпре уклањају сви подаци из свих табела базе Библиотека на локалном серверу, а потом се табеле попуњавају подацима који ће служити као модел за написане упите у овом раду.

3.5 Упити над подацима у бази података

Упити над базом података се у програмском језику Пајтон извршавају на сличан начин као и SQL наредбе које су приказане у претходним примерима.

Као што је већ показано, сваки курсор је направљен над везом са базом података и потребно је позвати методу `execute` и проследити јој одговарајућу ниску са SQL упитом.

За разлику од радних окружења за рад са MySQL базама података код којих се резултат упита одмах приказује, позивањем методе `execute` у програму написаном у програмском језику Пајтон, сви резултати биће смештени у радну меморију рачунара све док их не прочитамо. Због тога је потребно обратити пажњу на потенцијални обим резултата који се покретањем упита могу добити, па се због одговорног коришћења системских ресурса, број резултата може ограничити коришћењем клауза `LIMIT`, `HAVING` или додатним условима рестрикције у SQL упитима.

Резултате упита можемо да прочитамо коришћењем метода `fetchmany` и `fetchall`. Метода `fetchmany(size = k)` чита следећих `k` редова резултата упита и враћа листу уређених торки програму. Природан број `k` се функцији прослеђује помоћу аргумента `size`. Уколико до краја резултата упита има мање од `k` редова сви редови ће бити прочитани, а уколико нема више редова биће враћена празна листа. Метода `fetchall()` чита све редове резултата до краја и враћа листу уређених торки програму, а уколико нема редова у резултату враћа празну листу. Потребно је прочитати све редове текућег резултата како би над истим курсором могао да се изврши неки нови упит.

У програмском сегменту приказаном у листингу 3.9 биће приказане наредбе којима се издвајају називи свих табела у бази података уз проверу да ли је резултат упита празан. Резултат рада овог програма је листа торки чији су једини чланови називи табела, који се и исписују у једном реду стандардног излаза.

```
kursor = veza.cursor()
upit_tabele = "SHOW TABLES"
try:
    kursor.execute(upit_tabele)
    rezultati = kursor.fetchall()
    if(len(rezultati) == 0):
        print("Nema tabela u bazi podataka")
    else:
        print("Tabele u bazi Biblioteka su: ")
        for r in rezultati:
            print(r[0], end = " ")
        print("")
except mysql.connector.Error as e:
    print(e.msg)
kursor.close()
veza.close()
```

Листинг 3.9: Издвајање назива свих табела у бази података

Прикажимо један од начина за решавање проблема пребројавања текућих и завршених позајмица у бази података **Библиотека**. У листингу 3.10 је приказан MySQL упит којим се издвајају шифре чланова и укупан број различитих књига које је члан прочитао или тренутно чита, сортирано опадајуће према броју прочитаних књига.

```
with sve_pozajmice as(
    SELECT SIFC, SIFK FROM drzi
    UNION
    SELECT SIFC, SIFK FROM pozajmica
) SELECT SIFC, COUNT(SIFK)
FROM sve_pozajmice
GROUP BY SIFC
ORDER BY 2 DESC
```

Листинг 3.10: Употреба привремених табела у упитима

Приметимо да је у овом упиту искоришћена директива **with** чиме је направљена привремена табела унутар упита (енг. *Common Table Expression*).

Уколико претходни упит доделимо као ниску променљивој `upit_pozajmice` и потом извршимо упит сви редови резултата биће сачувани у радној меморији. Ако желимо да истакнемо прва три корисника тако што ћемо их приказати са бројем прочитаних књига, сваког у посебном реду, а да шифре других корисника прикажемо у једном реду то можемо да урадимо на начин приказан у листингу 3.11.

```
try:
    kursor.execute(upit_pozajmice)
    rezultati=kursor.fetchmany(size = 3)
    i=1
    if(len(rezultati) > 0):
        for r in rezultati:
            print(str(i)+".", r[0],r[1])
            i += 1
    rezultati = kursor.fetchall()
    if(len(rezultati)>0):
        for r in rezultati:
            print(r[0], end = " ")
        print("")

except mysql.connector.Error as e:
    print(e.msg)

kursor.close()
veza.close()
```

Листинг 3.11: Извршавање упита у програмском језику Пајтон

Приметимо да су прва три резултата упита прочитани помоћу методе `fetchmany`, док су преостали резултати прочитани помоћу методе `fetchall`. Уколико се до краја извршавања програма, након позивања методе `fetchmany` не прочитају сви резултати упита приказаће се грешка:

```
mysql.connector.errors.InternalError: Unread result found
```

Како би се ова грешка избегла, уколико резултат читања није била празна листа, на крају програма се може позвати метода `fetchall`.

Сортирање, које је реализовано у SQL упиту претходног примера, могло је да се реализује и у програмском језику Пајтон. Уколико резултат упита садржи велики број торки ефикасније је искористити сортирање у упиту због бољих перформанси база података при раду са великом количином података. Поред тога, уколико се сортирање обави у упиту, број редова резултата може бити редукован на неопходан број редова употребом клауза **LIMIT** и **HAVING**, што може смањити број редова резултата који се преноси у радну меморију рачунара [25].

У потпоглављу 3.4 приказан је пример једног уноса података код кога је целобројна вредност, која је била у саставу примарног кључа, била непозната. Програмер би приликом сваког уноса података морао да одреди тренутну највећу вредност примарног кључа у некој табели базе података и потом да је ручно уноси у програму. Једно од могућих решења овог проблема могло би да буде писање функције у програмском језику Пајтон која ће приликом сваког новог уноса одредити прву наредну вредност индекса за следећи унос података. Пример имплементације такве функције приказан је у листингу 3.12.

```
def slobodna_sifc(kursor):
    upit = "SELECT MAX(SIFC) FROM clan"
    try:
        kursor.execute(upit)
        rezultati = kursor.fetchall()
        return int(rezultati[0][0]) + 1
    except mysql.connector.Error as e:
        print(e.msg)
```

Листинг 3.12: Одређивање првог слободног индекса у табели CLAN

Вредности у торкама које се добијају читањем помоћу метода **fetchmany** и **fetchall** су ниске без обзира на тип података у бази података. Уколико желимо да их користимо у програму као број или неки други тип података потребно је да их конвертујемо у пожељни тип позивањем адекватних метода и функција.

Наведени примери јасно показују да су све кључне функционалности упитног језика MySQL доступне и у програмском језику Пајтон што омогућава програмеру да користи предности базе података у програмима писаним у програмском језику Пајтон. У поглављу 5, биће приказана имплементација апликације за рад са биолошким подацима смештеним у бази података.

Глава 4

Преглед биоинформатичких појмова

4.1 Обрада биолошких података

Биоинформатика је научна област која се интензивно развија од друге половине двадесетог века. Холандски научници Паулијен Хогеверг и Бен Хеспер су међу првима користили термин *биоинформатика* како би описали проучавање информатичких процеса у биолошким системима [26]. На развој биоинформатике повољно су утицали напредак у секвенцирању дезоксирибонуклеинске и рибонуклеинске киселине (скраћено ДНК и РНК), повећање капацитета за складиштење и обраду података, међународни пројекти попут Пројекта људског генома (енг. *Human Genome Project*) и примена метода машинског учења и вештачке интелигенције у раду са биолошким подацима [27].

Како би могли да раде са биолошким подацима биоинформатичари користе различите софтверске алате од којих су неки доступни и путем интернета. Помоћу њих могу да се преузму, анализирају, обрађују и упоређују секвенце молекула ДНК, РНК и протеина. Због разноврсне природе биолошких података, не постоји јединствена база која обухвата све биолошке податке. Међутим, доступан је све већи број база података са биолошким подацима којима може да се приступи путем интернета. Већина њих садржи бесплатне информације доступне академским корисницима, али постоје и сајтови који за приступ бази података захтевају претплату. Уз ресурсе на овим сајтовима могу да се пронађу и детаљни описи клиничког стања, генетских мутација, а омогућена је и претрага базе гена на основу прослеђених ДНК и РНК секвенци [27].

Развој софтверских алата који обрађују све веће количине биолошких података омогућава бољу анализу и разумевање функционисања организама у здравом и болесном стању и доприноси у истраживању животних процеса на молекуларном нивоу. Због тога биоинформатика, заједно са напретком у биологији и медицини, има све већи утицај на различите аспекте људског живота и здравља [28].

У овом поглављу рада говорићемо о поравнању нуклеотидних секвенци и филогенетској анализи, као важним поступцима у савременој биоинформатици. Како би могло да се говори о овим поступцима уведемо детаљније појмове ДНК и РНК.

Основни градивни елементи нуклеинских киселина (ДНК и РНК) су нуклеотиди. Нуклеотиди представљају фосфатне естере нуклеозида, који садрже шећер (пентозу) повезану гликозидном везом са азотном базом. Азотне базе могу бити пуринске (аденин, гуанин) или пиримидинске (цитозин, тимин/урацил). Нуклеотиди се повезују фосфодиестарским везама у полинуклеотидне ланце, који чине молекуле ДНК и РНК. Поступак одређивања редоследа нуклеотида у молекулу ДНК и РНК зове се *секвенционирање* [29].

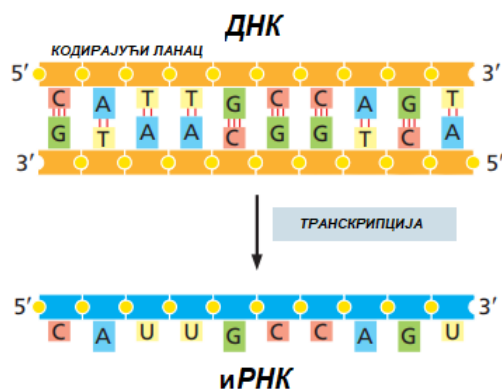
Код живих организама ДНК се састоји од два полинуклеотидна ДНК ланца. Сваки нуклеотид чине шећер - дезоксирибоза повезана са фосфатном групом, док азотна база може да буде аденин (А), цитозин (С), гуанин (G) и тимин (Т). ДНК ланци се простиру антипаралелно - усмерени су у супротним правцима, а повезани су водоничним везама. Притом важи *принцип комплементарности*: аденин једног ДНК ланца је увек у пару са тимином другог ДНК ланца, а гуанин са цитозином. Нуклеотиди у оквиру ДНК ланаца повезани су везама између шећера и фосфата (фосфодиестарским везама) и њихов редослед представља основу кодирања генетског материјала. Ове везе омогућавају ротацију сваког пара база за 36° у односу на суседне парове у ДНК ланцима чиме молекула ДНК добија облик двоструке спирале (енг. *double helix*). Молекула РНК се код живих организама састоји од једног полинуклеотидног ланца. У односу на молекула ДНК, због своје структуре молекула РНК је нестабилнији, што му омогућава да учествује у различитим ћелијским процесима. Сваки нуклеотид чине шећер - рибоза повезана са фосфатном групом, док су све азотне базе исте као код молекула ДНК осим тимина (Т), који је замењен са урацилом (U)[29].

Код честица вируса, ДНК и РНК могу бити и једноланчани и дволанчани молекули. Неки вируси користе РНК за чување свог генетског материјала.

На тај начин они успевају да по уласку у ћелију домаћина убрзају процес синтезе вирусних протеина. Део ДНК ланца (или РНК ланца код неких вируса) који садржи информацију за синтезу протеина или РНК зове се *ген*.

Пример вируса са дволанчаним молекулом РНК је ротавирус из породице вируса *Reoviridae*. Због лаког начина преношења ротавирусима се инфицирају готово сва деца до своје треће године живота [30].

У биоинформатици се и ДНК и РНК молекули приказују као ниске азотних база при чему се, у случају да су молекули били дволанчани, користи само један ланац нуклеотида - кодирајући ланац (енг. *sense strand*). Кодирајући ланац представља онај ланац који има идентичан распоред нуклеотида као информациона РНК (енг. *messenger RNA*), код које је у сваком нуклеотиду тимин (Т) из кодирајућег ланца замењен урацилом (U). Информациона РНК се краће означава са иРНК. Ланац иРНК представља ланац нуклеотида који настаје у процесу транскрипције и служи за пренос информација из ДНК до рибозома, где се та информација даље користи за синтезу протеина. Илустрација процеса транскрипције приказана је на слици 4.1. Уколико је молекул био дволанчани, други ланац се може једнозначно реконструисати захваљујући принципу комплементарности [29].



Слика 4.1: Приказ процеса транскрипције [29]

У раду ће бити коришћене нуклеотидне секвенце различитих сојева вируса SARS-CoV-2, који је изазвао пандемију 2020. године. Овај вирус користи једноланчане молекуле РНК за свој генетски материјал како би, одмах по инфицирању, ћелија домаћина започела синтезу вирусних протеина чиме се брже проширује инфекција.

Нуклеотидне секвенце коришћене у раду преузете су из базе гена Националног центра за биотехнолошке информације - NCBI (енг. *National Center for Biotechnology Information*) у FASTA формату, након чега су смештене у базу података. Више о самом поступку припреме секвенци и њиховом смештању у базу података биће речено у поглављу 5.

4.2 Поравнавање нуклеотидних секвенци

Поравнавање секвенци представља једну од основних метода у биоинформатици, која омогућава упоређивање биолошких секвенци, попут ДНК, РНК или протеинских секвенци. Циљ ове технике је да се пронађу делови у којима се секвенце поклапају, што може да указује на њихово заједничко еволутивно порекло или сличну структуру и функцију. У овом поглављу биће приказани основни концепти који се односе на поравнавање нуклеотидних секвенци ДНК и РНК ланаца у биоинформатици. Поред тога биће описан начин рада програма Clustal Omega, који је коришћен у овом раду.

4.2.1 Вишеструко поравнавање нуклеотидних секвенци и функција циља

Као што је већ речено у поглављу 4.1 основна улога ДНК (и РНК код неких вируса) је чување информација за синтезу протеина. Током еволуције, ДНК се мења услед мутација, које најчешће подразумевају замене (супституције), брисање (делеције) или убацивање (инсерције) једног или више нуклеотида у ДНК ланцу. Поред симбола који представљају ознаке азотних база и ознаке за неодређену азотну базу N, код поравнања нуклеотидних секвенци користи се и посебан симбол „—“ (*симбол празнине*), који означава одсуство нуклеотида и служи како би се добило боље поравнавање. *Проширена нуклеотидна секвенца* настаје када се у оригиналну нуклеотидну секвенцу унесу један или више симбола празнине. Симбол празнине се може унети на почетак, крај или између било која два карактера нуклеотидне секвенце [31].

Глобално поравнавање две нуклеотидне секвенце подразумева да се две проширене нуклеотидне секвенце поравнају тако да сваки карактер једне нуклеотидне секвенце одговара тачно једном карактеру друге нуклеотидне секвенце, али при томе два празна симбола не смеју бити поравната једно с другим [31].

Пример једног глобалног поравнања секвенци ССААСТТТЦГАТСТСТТТГТ АГАТСТ и ААСАСТАСАСАСССТСТТ је:

ССААСТТТЦГАТСТСТТГТАГАТСТ
--ААС-ТАСАСАСССТСТТ-----

Локално поравнање се односи на поравнање одређених сегмената две нуклеотидне секвенце, при чему се такође избегава поравнавање празних симбола једних са другима [31].

Пример једног локалног поравнања секвенци ССААСТТТЦГАТСТСТТТГТ АГАТСТ и ААСАСТАСАСАСССТСТТ је:

АТ-СТСТТ
АСССТСТТ

Број различитих могућности за глобална и локална поравнања две секвенце је велики, па се због избора оптималног поравнања дефинише функција којом се одређује успешност (енг. *score*) поравнања [31].

Алгоритми за поравнање нуклеотидних секвенци могу да се, према броју нуклеотидних секвенци које учествују у поравнању, поделе на алгоритме за поравнање у паровима и вишеструко поравнање нуклеотидних секвенци (енг. *Multiple Sequence Alignment*), где у поравнању могу да учествују и више од две нуклеотидне секвенце.

Проблем вишеструког поравнања секвенци формулише се као задатак у коме је, за дати скуп од k секвенци дужине највише n , потребно одредити оптимално поравнање свих секвенци. *Поравнање* k нуклеотидних секвенци S_1, S_2, \dots, S_k дефинише се као матрица карактера:

$$A = (A_{ij}) \quad \text{за} \quad 1 \leq i \leq k, \quad 1 \leq j \leq l,$$

где сваки ред i представља проширење секвенце S_i (у ознаци S'_i) дужине l (са уметнутим симболима празнина), а свака колона j одговара поравнатој позицији. Циљ је одредити матрицу поравнања A која минимизује неку задату функцију сличности или растојања између секвенци [32].

Проблем вишеструког поравнања нуклеотидних секвенци је NP-комплетан проблем. То значи да не постоји познати алгоритам који у полиномијалном времену може да реши све случајеве овог проблема па се у пракси развијају хеуристичке и апроксимационе методе за решавање овог проблема [32].

Функција која се користи у вишеструком поравнању секвенци за оцену поравнања зове се *функција циља* (енг. *cost function*). Нека је дат скуп од k секвенци S_1, S_2, \dots, S_k , и нека је дата матрица поравнања A са k редова и l колона. Функција циља w дефинише се као функција над проширеном азбуком (ознаке азотних база и симбол празнине) и одређује укупан трошак једне колоне у матрици поравнања (за нуклеотидне секвенце колона у матрици поравнања A представљена је k -торком). Укупан трошак поравнања A са l колона дефинише се као:

$$D(A) = \sum_{j=1}^l w(A_{1j}, A_{2j}, \dots, A_{kj}),$$

при чему је са A_{ij} означен карактер на j -тој позицији i -те нуклеотидне секвенце S'_i у матрици поравнања A [33].

4.2.2 Програм Clustal Omega

Clustal Omega (Clustal Ω) представља савремени програм за вишеструко поравнање нуклеотидних и протеинских секвенци. Развијен је као нова верзија програма ClustalW и ClustalX тако да поред значајног унапређења у ефикасности, тачности и скалабилности, омогући ефикасно поравнање чак стотине хиљада секвенци [34].

Прве верзије програма Clustal су користиле једноставан метод за конструисање водећих стабала (енг. *guide trees*). Водеће стабло представља упутство на основу кога се одређује редослед поравнања код вишеструког поравнања секвенци. Конструкција водећег стабла имала је временску сложеност $O(N^2)$ за N улазних секвенци, што представља значајно ограничење у раду са великим бројем секвенци [34].

Код програма Clustal Omega процес поравнања секвенци започиње одређивањем k -мер профила сваке секвенце (пребројавањем подниски дужине k). Добијени резултати се користе за векторску репрезентацију сваке секвенце. Потом се конструише матрица растојања (енг. *distance matrix*) на основу репрезентација секвенци. Ова матрица користи се даље за конструкцију водећег стабла. Програм Clustal Omega за конструкцију водећег стабла користи технику mBed која, за разлику од раније коришћених метода, има временску сложеност $O(N \log(N))$ што убрзава рад са великим скуповима секвенци. Код ове технике се, уместо одређивања растојања свих секвенци по паровима, за сваку секвенцу одређује растојање у односу на мали број референтних секвенци. На основу тих растојања врши се груписање секвенци према сличности што значајно утиче на ефикасност конструкције водећег стабла [35]. Више детаља о овој методи може се пронаћи у раду [36].

Још једно значајно унапређење програма Clustal Omega у односу на претходне верзије, односи се на употребу механизма за поравнавање који не користи класичне алгоритме динамичког програмирања. Clustal Omega помоћу веома прецизног алата HAlign, користећи водеће стабло, поравнава профиле скривених Марковљевих модела (енг. *Hidden Markov Models*) што значајно повећава његову прецизност у односу на претходне верзије [35]. Више о поравнању профилних скривених Марковљевих модела може се пронаћи у раду [37].

Clustal Omega представља програм отвореног кода (енг. *open-source program*) и објављен је 2011. године. Прве верзије овог програма су подржавале искључиво рад са протеинским секвенцама [34].

Већ 2012. године услед првог већег ажурирања, омогућено је поравнавање ДНК и РНК нуклеотидних секвенци. Још од прве верзије подржане су улазне датотеке у FASTA, CLUSTAL, MSF, PHYLIP, SELEX, STOCKHOLM и VIENNA формату. Излазне датотеке су датотека са поравнањем секвенци (најчешће датотека са екстензијом *.aln*), датотека са водећим стаблом и датотека са матрицом растојања. Програм **Clustal Omega** се покреће из командне линије (енг. *command-line*). Ова верзија је доступна за све водеће оперативне системе. Поред тога је доступан и путем онлајн сервиса, који омогућава његово коришћење без инсталације [34].

У оквиру овог рада коришћена је верзија програма **Clustal Omega**, која се покреће из командне линије и која може да се преузме путем линка [38]. Више о подешавању и начину покретања програма биће речи у поглављу 5.5. Резултати поравнања послужили су као основа за даљу филогенетску анализу о којој ће бити речи у следећем сегменту овог рада.

4.3 Филогенетска анализа

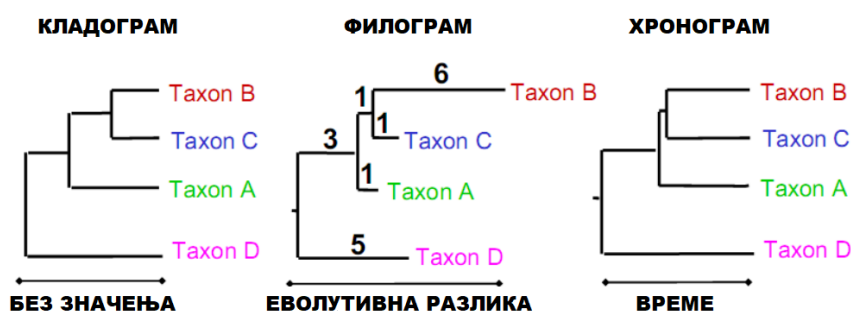
Филогенетика (енг. *phylogenetics*) представља научну дисциплину која се бави проучавањем еволутивних односа између организама и њихових гена анализом информација заснованих на поравнању секвенци ДНК, РНК и протеина. Популације се еволутивно мењају помоћу различитих мутација. Неке од мутација које настају код појединих јединки постају карактеристичне за све њене потомке. Поравнавањем секвенци савремених организама могу да се идентификују обрасци и сличности са неким организмима из прошлости. На основу тих образаца могу се конструисати стабла која приказују раздвајање врста током времена [39]. У овом поглављу рада говорићемо о основним концептима филогенетске анализе. Поред тога биће описан начин рада програма **FastTree** који је коришћен за конструкцију филогенетског стабла у раду.

4.3.1 Филогенетска стабла

Филогенетска стабла представљају специјалну врсту повезаних усмерених ацикличких графова. У контексту молекуларне филогенетике, листови (енг. *leaves*) филогенетског стабла представљају посматране секвенце, односно појединачне гене, популације или врсте [31].

У филогенетским стаблима, организми или групе организама чије се секвенце пореде и који се налазе у листовима зову се *таксони*. Унутрашњи чворови (енг. *internal nodes*) одговарају претпостављеним заједничким прецима таксона. Гране одређују еволутивне односе између секвенци, указујући на правац наслеђивања и степен сродности. Стабло може да има корен (енг. *rooted*), када постоји заједнички предак свих таксона, или да нема корен (енг. *unrooted*), када се приказују само односи, без указивања на правац еволуције [40].

У зависности од значења које се приписује дужини грана разликују се три врсте филогенетских стабала: *клатограм*, *филограм* и *хронограм*. Филогенетско стабло је *клатограм* ако дужине његових грана немају никакво значење. Клатограм приказује само односе између таксона без информације о степену еволутивне и временске разлике. *Филограм* представља врсту филогенетског стабла код кога дужине грана одговарају степену еволутивних разлика између таксона. Филогенетско стабло код кога дужине грана представљају време протекло од заједничког претка до таксона зове се *хронограм* (*ултраметричко стабло*). За разлику од филограма код кога листови не морају бити на истој висини (због различитог степена еволутивних промена), код хронограма су сви таксони углавном на истој удаљености од корена (осим ако се не ради о секвенцама изумрлих организама) [41]. Врсте филогенетских стабала приказане су на слици 4.2.



Слика 4.2: Врсте филогенетских стабала [41]

Методe за конструкцију филогенетских стабала могу се поделити у две категорије: *методe засноване на растојањима* и *методe засноване на карактерима*. Методe засноване на растојањима одређују растојања између поравнања по паровима, а потом на основу матрице еволутивних растојања конструишу стабло. За разлику од њих, методe засноване на карактерима анализирају положаје нуклеотида у поравнању, претражују простор свих могућих стабала и оцењују које најбоље одговара подацима [41].

4.3.2 Програм FastTree

Матрица растојања која представља резултат рада програма **Clustal Omega** није у потпуности погодна за даљу филогенетску анализу. Програм **Clustal Omega** приликом конструкције матрице растојања користи такозвана *p*-растојања (енг. *p-distances*), која представљају однос броја различитих нуклеотида и укупног броја нуклеотида у поравнању. На тај начин се потцењују вишеструке мутације кроз време и не узимају се у обзир вероватноће догађања различитих мутација. Поред тога симболи празнина или недостајућих азотних база могу бити занемарени током обраде [42][43]. Стога добијено вишеструко поравнање мора да буде прослеђено програму, као што је **FastTree**, који ће прецизније одредити еволутивну удаљеност посматраних узорака и конструисати филогенетско стабло.

Програм **FastTree** представља програм отвореног кода и иако се метод његовог рада често сврстава у методе максималне веродостојности (енг. *Maximum Likelihood*), он заправо представља хибридни приступ који комбинује елементе метода заснованих на растојању и метода заснованих на карактерима. Улазна датотека програма **FastTree** представља датотеку са поравнањем у једном од стандардних формата (FASTA или PHYLIP). Због тога је овај програм погодно користити уз програме за вишеструко поравнање секвенци као што је **Clustal Omega**. Резултат рада програма **FastTree** представља датотека са филогенетским стаблом у NEWICK формату [44].

У првој фази свог рада програм **FastTree** конструише почетно стабло користећи брзу верзију *Neighbour Joining* алгоритма. На тај начин конструише се структура стабла у чијим се унутрашњим чворовима налази расподела вероватноћа за сваки карактер у нуклеотидној секвенци. Потом се, у другој фази врши итеративна оптимизација добијеног стабла коришћењем метода за одређивање максималне веродостојности уз употребу једног од модела еволуције (енг. *evolutionary model*). Модели еволуције садрже вероватноће промена једног нуклеотида (или аминокиселине) другим. На тај начин се врши корекција добијених разлика између секвенци. Ова конструкција омогућава програму **FastTree** да конструише филогенетска стабла високе тачности, слична онима добијеним класичним методама, али у краћем временском року [45].

Мера поузданости груписања у филогенетском стаблу одређује се подршком чвора (енг. *node support*). Овај податак представља ниво поверења у исправност одређеног чвора у филогенетском стаблу и често се одређује бутстреп (енг. *bootstrap*) методом. Бутстреп метода се заснива на вишеструком узорковању података и одређивању броја појављивања грана при сваком узорковању [46].

Како би се побољшала ефикасност у раду са великим бројем узорака, програм **FastTree** користи алтернативну, бржу меру подршке засновану на SH тесту (енг. *Shimodaira–Hasegawa test*). За сваки чвор се одређује вероватноћа да је топологија стабла која садржи тај чвор боља од других топологија које настају мањим променама у стаблу (локална претрага у том делу стабла). Као резултат се добија реалан број између 0 и 1 при чему се вредности веће од 0,9 сматрају високим нивоом подршке, вредности између 0,7 и 0,9 умереним нивоом подршке, док се вредности мање од 0,7 сматрају slabим нивоом подршке и треба их узети са резервом приликом анализе добијеног стабла [47].

У оквиру овог рада коришћена је верзија програма **FastTree**, која се покреће из командне линије и која може да се преузме путем странице [44]. Више о подешавању и начину покретања овог програма биће речи у поглављу 5.5.

Глава 5

Развој апликације за рад са биолошким подацима

У овом поглављу биће речи о апликацији за рад са биолошким подацима који су сачувани у бази података. Апликација је направљена у програмском језику Пајтон, док је графички кориснички интерфејс развијен помоћу библиотеке *Tkinter*. *Tkinter* представља најстарију и најшире коришћену стандардну библиотеку за израду графичког корисничког интерфејса у програмском језику Пајтон. Подржава све стандардне компоненте за развој графичког корисничког интерфејса – дугмиће, текстуална поља, падајуће меније и многе друге [48].

Апликација обрађује биолошке податке, који су смештени у бази података, врши поравнање нуклеотидних секвенци узорака позивањем програма **Clustal Omega**, конструише филогенетско стабло позивањем програма **FastTree** и потом приказује резултате поравнања и филогенетско стабло. Поред рада са подацима из базе података, апликација омогућава поравнавање нуклеотидних секвенци из датотека и поређење свих одабраних узорака са једним изабраним узорком како би се одредио проценат њихове сличности. Додатне развијене функционалности апликације су унос, ажурирање, брисање и претраживање сојева вируса и узорака у бази података. Програмски код апликације уз пратећу документацију је доступан за слободно преузимање¹.

¹<https://github.com/borellebeg/MasterRad>

5.1 Вирус SARS-CoV-2

База података је направљена у програму XAMPP на начин описан у поглављу 3. У базу података смештени су подаци о различитим сојевима вируса SARS-CoV-2 који је изазвао пандемију COVID-19 током 2020. године. Овај вирус има висок степен генетске варијабилности, уз релативно честе мутације и настанак нових сојева, што је послужило као мотивација за филогенетску анализу и поравнавање нуклеотидних секвенци његових једанаест различитих сојева.

Постоји више различитих система означавања који се користе за праћење и класификацију сојева вируса као што је SARS-CoV-2. Сваки од њих се користи у различите сврхе међу којима су: научна истраживања, информисање јавности и генетска анализа. Четири најпознатија система коришћена од стране Светске здравствене организације (енг. *World Health Organization*) су PANGO, WHO, GISAID и NEXTSTRAIN. Сваки од поменутих система означавања се употребљава за означавање соја вируса на другачији начин [49].

PANGO номенклатура (енг. *Phylogenetic Assignment of Named Global Outbreak Lineages*) у својој ознаци прати порекло и развој вируса кроз линије и подлиније, као што су B.1.617.2 и C.37. Овај систем означавања је најпогоднији за научнике који прате мутације и генетске односе између сојева. Због тога је ова ознака одабрана за примарни кључ у табели са сојевима вируса [50][51].

Ознака коју додељује Светска здравствена организација означава се са WHO и представља једноставну и општеприхваћену ознаку која се користи зарад лакше комуникације у јавности, а и како би се избегла стигматизација земље у којој је вирус први пут откривен. Примери WHO ознака су: Alpha, Beta и Delta и додељују се углавном сојевима који имају значајне мутације у односу на постојеће сојеве и за које се претпоставља да имају већи потенцијал да утичу на даље ширење, дијагностику и имуни одговор инфицираних особа (енг. *variants of interest*) [51][52].

GISAID класификација се односи на велике групе вируса који имају неке сличне мутације, као што су G, GR или GRA. GISAID је такође и највећа светска база података за геномске секвенце вируса [50].

NEXTSTRAIN класификација користи ознаку године када је сој први пут узоркован и слова за ознаку врсте соја вируса чиме се лакше стиче увид у еволуцију вируса [50][51].

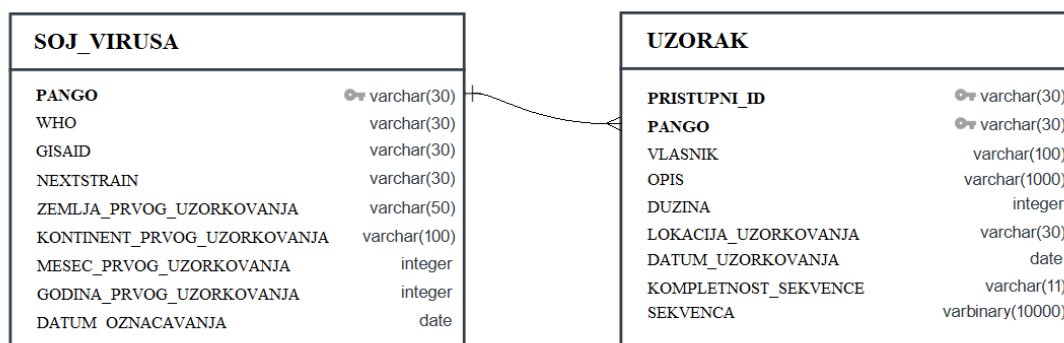
5.2 Структура базе података и прављење табела

База података коришћена за чување биолошких података потребних за рад апликације има једноставну структуру. Табела 5.1 садржи списак табела у бази података и кратак опис њихових атрибута.

назив табеле	преглед и опис атрибута
SOJ_VIRUSA	PANGO ознака соја вируса, ознака Светске здравствене организације, GISAID ознака, NEXTSTRAIN класификација, земља првог узорковања, континент првог узорковања, месец првог узорковања, година првог узорковања и датум означавања
UZORAK	идентификатор приступа и PANGO ознака соја вируса (страни кључ на табелу SOJ_VIRUSA), власник узорка, опис, дужина узорка, локација узорковања, датум узорковања, комплетност секвенце и секвенца

Табела 5.1: Преглед и опис атрибута у бази апликације

На слици 5.1 приказана је шема релационе базе података са типовима атрибута и означеним релацијама између табела. Примарни кључ је означен малом ознаком кључа поред типа атрибута.



Слика 5.1: Шема релационе базе података

Пример имплементације функције којом се у програмском језику Пајтон праве табеле `soj_virusa` и `uzorak` приказан је у листингу 5.1.

```
def kreiranje_tabela(baza):
    kursor = baza.cursor()
    sql = "DROP TABLE uzorak"
    kursor.execute(sql)
    sql = "DROP TABLE soj_virusa"
    kursor.execute(sql)
    sql = "CREATE TABLE soj_virusa ("
    sql+= "pango VARCHAR(30) NOT NULL,"
    sql+= "who VARCHAR(30),"
    sql+= "gisaid VARCHAR(30),"
    sql+= "nextstrain VARCHAR(30),"
    sql+= "zemlja_prvog_uzorkovanja VARCHAR(50),"
    sql+= "kontinent_prvog_uzorkovanja "
    sql+= "VARCHAR(100) NOT NULL,"
    sql+= "mesec_prvog_uzorkovanja INT,"
    sql+= "godina_prvog_uzorkovanja INT,"
    sql+= "datum_oznacavanja DATE,"
    sql+= "PRIMARY KEY (pango)) ENGINE=InnoDB"
    kursor.execute(sql)
    sql = "CREATE TABLE uzorak ("
    sql+= "pristupni_id VARCHAR(30) NOT NULL,"
    sql+= "pango VARCHAR(30) NOT NULL,"
    sql+= "vlasnik VARCHAR(100),"
    sql+= "opis VARCHAR(1000),"
    sql+= "duzina INT,"
    sql+= "lokacija_uzorkovanja VARCHAR(30),"
    sql+= "datum_uzorkovanja DATE,"
    sql+= "kompletnost_sekvence VARCHAR(11),"
    sql+= "sekvenca VARBINARY(10000) NOT NULL,"
    sql+= "PRIMARY KEY (pristupni_id),"
    sql+= "FOREIGN KEY (pango) "
    sql+= "REFERENCES soj_virusa(pango)) ENGINE=InnoDB"
    kursor.execute(sql)
```

Листинг 5.1: Функција којом се праве табеле soj_virusa и uzorak у бази података

5.3 Преузимање, припрема и унос биолошких података у базу података

У овом поглављу рада биће речи о FASTA формату и начину припреме нуклеотидних секвенци за унос у базу података.

5.3.1 FASTA формат за нуклеотидне секвенце

Као што је већ поменуто све нуклеотидне секвенце коришћене у раду преузете су из базе гена Националног центра за биотехнолошке информације (енг. *National Center for Biotechnology Information*) у FASTA формату. У FASTA формату, линија која се налази испред нуклеотидне секвенце зове се дефинициона линија и мора да почиње карактером (>), након чега следи јединствени идентификатор секвенце (SeqID). Идентификатор секвенце представља јединствену ознаку која не садржи белине и његова дужина није већа од двадесет пет карактера. У идентификатору су дозвољени карактери: слова, бројеви, као и специјални карактери (–, _, ., :, * и #). Након ознаке могу се налазити и информације о организму од кога потиче секвенца, научно име организма, а могу бити истакнуте и додатне карактеристике као што су сој вируса и хромозом. Након ових информација може се налазити и опис секвенце, као и карактеристика да ли је у питању комплетан геном или не. Дефинициона линија не сме да буде приказана у више редова односно садржи само једну ознаку за крај реда на свом крају (енг. *hard return*). У редовима после дефиниционе линије налази се нуклеотидна секвенца код које се у сваком реду налази највише осамдесет карактера. При том се, за означавање база, користе искључиво ознаке Међународне уније за чисту и примењену хемију (енг. *International Union of Pure and Applied Chemistry*) док се за неодређене азотне базе користи карактер N [53]. На слици 5.2 је приказан почетак једне од FASTA датотека које су коришћене у раду. Датотеке FASTA формата су због своје једноставности и јасно дефинисаног формата изузетно погодне за аутоматску обраду података.

```
>MZ388000.1 Severe acute respiratory syndrome coronavirus 2 isolate SARS-CoV-2/...
ACATTCGATCTCTGTAGATCTGTTCTCTAAACGAACCTTTAAAATCTGTGTGGCTGTCACTCGGCTGCAT
GCTTAGTGCACTCAGCAGTATAATTAATAACCTAATTACTGTCGTTGACAGGACACGAGTAACCTCGTCTA
TCTTCTGCAAGGCTGTTACGGTTTCGTCCGTGTTGCAGCCGATCATCAGCACATCTAGGTTTTGTCCGGG
TGTGACCGAAAGGTAAGATGGAGAGCCTTGTCCCTGGTTTCAACGAGAAAACACACGTCCAACCTCAGTTT...
```

Слика 5.2: Приказ садржаја FASTA датотеке

5.3.2 Преузимање и припрема података за унос

Подаци о сојевима вируса преузети су са званичне презентације Светске здравствене организације [49]. Сви подаци о сојевима смештени су у посебне редове датотеке `sojevi_virusa.txt` у редоследу који одговара атрибутима табеле `soj_virusa` приказане на шеми релационе базе података 5.1. Вредности атрибута раздвојене су карактером тачка и зарез. На слици 5.3 приказан је садржај датотеке `sojevi_virusa.txt`.

```
B.1.1.7 ;Alpha ;GR/501Y.V1;20I (V1) ;Ujedinjeno Kraljevstvo;Evropa;...
B.1.351 ;Beta ;GH/501Y.V2 ;20H (V2);NULL;Afrika;5;2020;18.12.2020
P.1 ;Gamma ;GR/501Y.V3 ;20J (V3);Brazil;Južna Amerika;11;2020;11.1.2021
B.1.617.2 ;Delta ;G/478K.V1 ;21A;Indija;Azija;10;2020;4.4.2021
B.1.427;Epsilon ;GH/452R.V1 ;21C ;SAD;Severna Amerika;3;2020;5.3.2021
P.2 ;Zeta ;GR/484K.V2;20B/S.484K ;Brazil;Južna Amerika;4;2020;17.5.2021
B.1.525 ;Eta ;G/484K.V3 ;21D;Ujedinjeno Kraljevstvo/Nigerija;Evropa...
P.3 ;Theta ;GR/1092K.V1 ;21E ;Filipini;Azija;1;2021;24.3.2021
B.1.526 ;Iota ;GH/253G.V1;21F ;SAD;Severna Amerika;11;2020;24.3.2021
B.1.617.1 ;Kappa ; G/452R.V3 ;21B ;Indija;Azija;10;2020;4.4.2021
C.37;Lambda;GR/452Q.V1;20D;Peru;Južna Amerika;8;2020;14.6.2021
```

Слика 5.3: Приказ садржаја датотеке `sojevi_virusa.txt`

Узорци и подаци о узорцима су преузети из базе података Националног центра за биотехнолошке информације у FASTA формату и њихове датотеке су смештене у директоријум UZORCI. Узорци коришћени у раду имају дуге низове нуклеотидних секвенци те је потребно одредити најоптималнији начин за њихово чување у бази података. Како се над секвенцама у бази неће вршити претрага, а и како би се убрзало њихово читање из базе података и смањила употреба меморије потребно је да се нуклеотидне секвенце пре уноса у базу података компресују. Нуклеотидне секвенце су најпре енкодиране у бинарни формат позивањем методе `encode()`. Ово омогућава њихову компресију позивањем функције `compress` из библиотеке `zlib`. На тај начин се постиже просечна уштеда меморије од чак 70, 72% за секвенце коришћене у раду. Добијени резултат има велики значај уколико се у бази података чува велики број нуклеотидних секвенци узорака.

За чување нуклеотидних секвенци у бази података одабран је тип `VARBINARY` који се користи за чување бинарних података променљиве дужине. Приликом читања секвенци из базе, секвенце прво морају да се декомпресују позивањем функције `decompress` из библиотеке `zlib`, а потом да се преведу из бинарног у текстуални формат позивањем методе `decode()`.

У табели 5.2 приказани су подаци о узорцима који су коришћени у раду и дужини њихових нуклеотидних секвенци пре и после компресије.

ознака узорка	А	В	%
MZ389980	29763	8829	70,336
MZ379528	29764	8724	70,689
MZ348330	29833	8842	70,362
MW932027	29817	8831	70,383
MZ384030	29810	8836	70,359
MZ375761	29878	8792	70,574
MZ388000	29730	8809	70,370
MZ368550	29777	8785	70,497
MZ381502	29813	8452	71,650
MZ384088	29800	8315	72,097
MZ385752	29834	8760	70,638

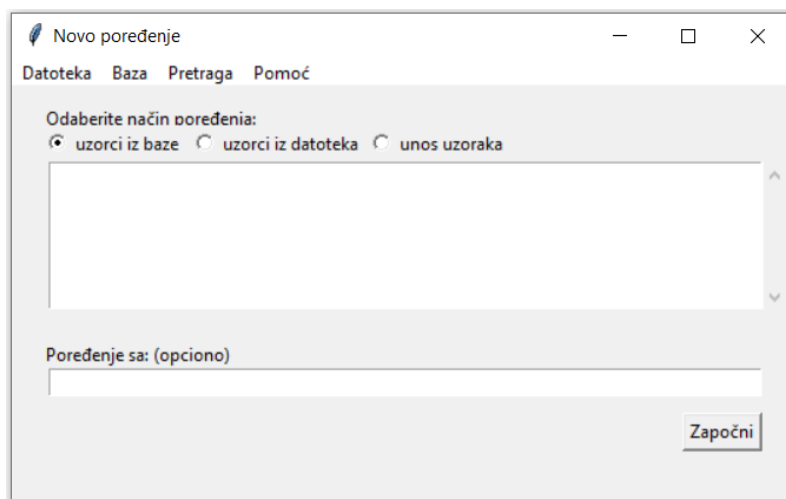
Табела 5.2: Дужина нуклеотидних секвенци узорака пре (А) и после компресије (В) и проценат уштеде меморије (%)

Функција којом се подаци о сојевима вируса и узорцима уносе у базу података је `unos_podataka (baza, direktorijum, infodat, infouzor)`. Овој функцији прослеђују се веза са базом података, путања ка директоријуму са FASTA датотекама узорака, путања ка текстуалној датотеци са подацима о сојевима вируса и путања ка текстуалној датотеци са подацима о узорцима (без нуклеотидних секвенци). Због своје структуре приказ ове функције у раду није могућ, али она чини део документације рада и може се прегледати у датотеци `baza.py`.

5.4 Графички кориснички интерфејс

Графички кориснички интерфејс је направљен помоћу библиотеке *Tkinter*. Цео пројекат је подељен у неколико датотека. Изворни код главног програма налази се у датотеци `main.py`. У главном програму је направљен објект класе `Applikacija`, чији се изворни код налази у датотеци `aplikacija.py`. У конструктору класе `Applikacija` се иницијализује главни прозор графичког корисничког интерфејса апликације. На почетку се позива конструктор наткласе `Tk` чиме се обезбеђује да главни прозор наслеђује сва својства прозора у библиотеци *Tkinter*. Све странице апликације представљају посебне класе које наслеђују класу оквира (класа *Frame*).

Странице графичког корисничког интерфејса су смештене на исту позицију тако да прекривају једна другу, при чему је у једном тренутку активан само један оквир. На почетку се, позивањем функције `show_frame(f1)` приказује прва страница интерфејса, приказана на слици 5.4



Слика 5.4: Почетна страница апликације

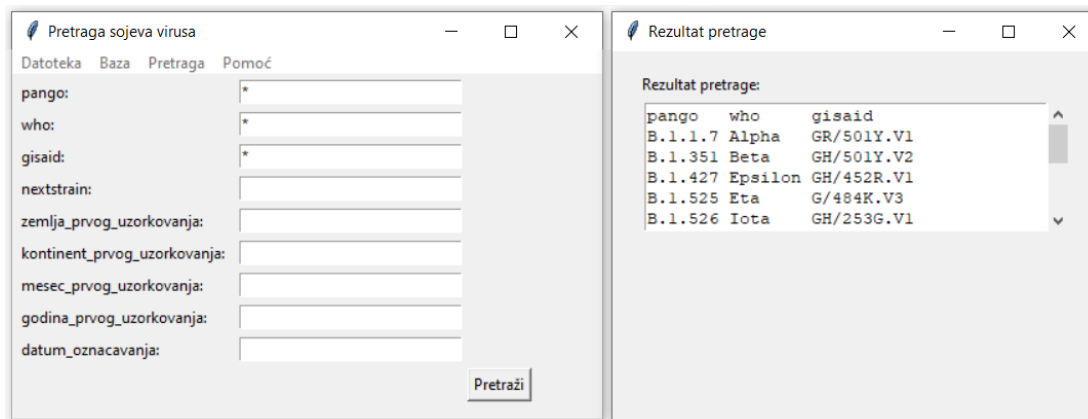
Инстанце свих страница су смештене у структуру података речник и страницама може да се приступа помоћу опција менија (класа *Menu*), који се приказује на врху свих страница. Мени садржи опције за рад са датотекама, унос, ажурирање, брисање и претрагу података у бази података као и странице за помоћ и информације о апликацији. У листингу 5.2 приказано је додавање ставки у подмени (енг. *submeni*) операције, а потом се подмени са ставкама додаје као падајућа листа *Datoteka* менија *meni*.

```
operacije = Menu(meni)
operacije.add_command(label="Novo poređenje",
                       command=self.novoPoredjenje)
operacije.add_command(label="Otvori konfiguraciju",
                       command=self.otvori)
operacije.add_command(label="Sačuvaj konfiguraciju",
                       command=self.sacuvajKonfiguraciju)
operacije.add_command(label="Otvori izveštaj",
                       command=self.otvoriIzvestaj)
meni.add_cascade(label="Datoteka", menu=operacije)
```

Листинг 5.2: Додавање ставки у подмени *Datoteka*

У класи `Aplikacija` су, поред метода којима се приказују одговарајуће странице, дефинисане и методе које обрађују акције и функционалности свих страница апликације.

За приказ резултата поравнавања нуклеотидних секвенци и резултате претраге базе података користе се помоћни прозори који се могу направити као инстанце класе `Toplevel`. Додатни прозор за резултате претраге, о којој ће бити речи у поглављу 5.5, приказан је на слици 5.5. Помоћу додатних прозора могуће је приказати и више резултата поређења што олакшава анализу резултата .



Слика 5.5: Изглед главног и помоћног прозора

У листингу 5.3 приказан је програмски код којим се генеришу помоћни прозори у апликацији.

```
rez = tk.Toplevel(self)
rez.minsize(550, 300)
rez.title("Rezultat pretrage")
rez.geometry("+%d+%d" % (570, 20))
l1 = Label(rez, text="Rezultat pretrage: ")
l1.place(x=20, y=15)
t1 = tk.scrolledtext.ScrolledText(rez, height = 6,
                                   width = 60, wrap="none")
t1.place(x=25, y=40)
```

Листинг 5.3: Генерисање помоћних прозора у апликацији

Изворни код класа које представљају странице апликације налази се у датотеци `stranice.py` док су изворни кодови функција које су везане за базу података у датотеци `baza.py`.

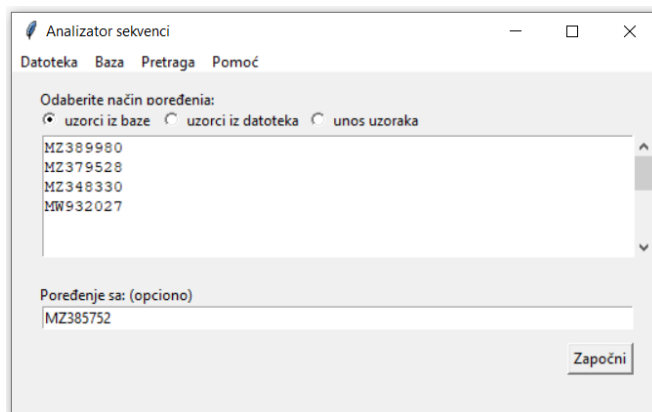
5.5 Преглед функционалности апликације

У овом поглављу рада биће представљене основне функционалности апликације.

5.5.1 Поравнавање нуклеотидних секвенци

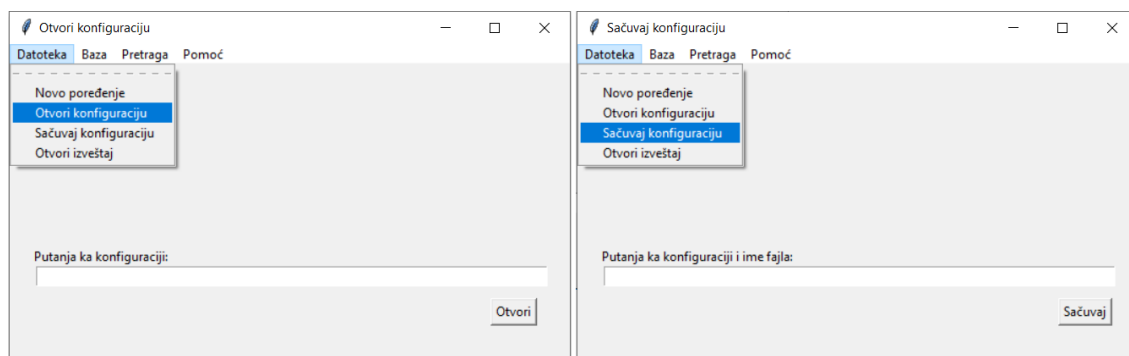
Одабир узорака чије ће нуклеотидне секвенце учествовати у поравнању омогућен је на почетној страници, која је приказана на слици 5.4. Могућ је одабир узорака који се налазе у бази података, узорака који су сачувани у датотекама и ручни унос узорака. Избор између три понуђене опције реализован је употребом радио дугмади (класа *RadioButton*). У случају да се користе узорци из базе података потребно је у текстуално поље (класа *ScrolledText*) унети називе узорака који ће учествовати у поравнању или оставити поље празно уколико сви узорци из базе треба да учествују у поравнању. Уколико се одабере опција за коришћење узорака који су сачувани у датотекама потребно је у текстуалном пољу навести путање ка свим FASTA датотекама са узорцима, сваку у посебном реду. Опција ручног уноса узорака омогућена је за краће нуклеотидне секвенце и потребно је унети назив узорка у једној линији текстуалног поља, а у следећој нуклеотидну секвенцу тог узорка. Пример исправно попуњене почетне странице приказан је на слици 5.6.

У сва три случаја омогућено је поређење једног истакнутог узорка са свим осталим узорцима. Унос назива, путање ка датотеци са узорком или ручни унос тог узорка врши се путем посебног текстуалног поља (класа *Entry*) у зависности од изабране опције за начин поређења.



Слика 5.6: Пример исправно унетих података о узорцима

У менију у картици **Datoteka** постоје опције за чување и отварање почетних конфигурација. Странице за отварање почетне конфигурације из датотеке, као и чување почетне конфигурације су приказане на слици 5.7. Ова функционалност скраћује време за унос назива и путања узорака код поравнања у којима учествује више узорака.



Слика 5.7: Странице за отварање и чување почетних конфигурација

Након унетих података, кликом на дугме **Započni** на почетној страници, врши се провера унетих података, а потом се отвара датотека **unaligned.fasta** за упис нуклеотидних секвенци.

Датотека **unaligned.fasta** садржи све нуклеотидне секвенце које учествују у поравнању и биће, након припреме, прослеђена програму **Clustal Omega** као датотека са улазним подацима. Због тога њена структура мора да буде одговарајућа. Сваки узорак се уноси тако што се у једној линији уноси карактер **>** и назив узорака, а потом у следећем реду нуклеотидна секвенца без ознака за прелазак у нови ред.

У листингу 5.4 су приказане наредбе за читање узорака чији су називи наведени у текстуалном пољу. Узорци се, након што су прочитани из базе података, прво декомпресују, потом декодирају као што је раније објашњено у овом поглављу и на крају се уписују у датотеку **unaligned.fasta**.

Структура датотеке **unaligned.fasta** је иста као и у случају да је одабрана опција за унос узорака из датотека, или за ручни унос узорака. Код ових опција разликује се само начин читања и парсирање узорака.

```
baza = povezivanje_sa_bazom("localhost", "root",
                             "", "baza")

kursor = baza.cursor()
izlaz = open("unaligned.fasta", "w")
d1 = f.myText1.get("1.0", "end-1c")
d1=d1.split("\n")
for u in d1:
    sql = "SELECT pristupni_id, sekvenca FROM uzorak "
    sql+= "WHERE pristupni_id = '"+u+"' "
    res = kursor.execute(sql)
    uzorak = kursor.fetchall()
    if uzorak:
        dec=zlib.decompress(uzorak[0][1])
        seq = dec.decode()
        konf = konf + uzorak[0][0]+"\\n"
        izlaz.write(">"+uzorak[0][0]+"\\n"+seq+"\\n")
    else:
        text = "Uzorak "+u+ " nije pronađen.\\n"
        text+= "Nakon provere pokušajte ponovo"
        f.myLabel1.config(text)
    return -1
```

Листинг 5.4: Читање узорака и припрема датотеке unaligned.fasta

За покретање програма Clustal Omega коришћене су функције из библиотеке Biopython. Функција ClustalOmegaCommandline припрема командну линију за покретање програма Clustal Omega. Коришћене су следеће опције:

- `infile` - путања до улазне датотеке;
- `outfile` - путања до излазне датотеке (за чување поравнања);
- `guidetree_out` - путања до водећег стабла;
- `distmat_out` - путања до матрице растојања;
- `percentid` - ако ова је опција активирана у матрици растојања се приказују проценти сличности (уместо броја разлика);

- `outfmt` - формат излазне датотеке;
- `outputorder` - редослед узорака у излазном поравнању;
- `seqtype` - врста нуклеотидне секвенце;
- `threads` - број процесорских нити које ће бити коришћене;
- `force` - ако је ова опција активирана, код вишеструког покретања поравнања све излазне датотеке које већ постоје биће испражњене.

Функција `clustalomega_cline()` из исте библиотеке извршава припремљену наредбу из командне линије и као резултат враћа поруке за стандардни излаз и за стандардни излаз за грешке. У листингу 5.5 су приказане наредбе у програмском језику Пајтон којима се подешавања командна линија и покреће програм Clustal Omega.

```
in_file = "unaligned.fasta"
out_file = "aligned.fasta"
clustalomega_cline = ClustalOmegaCommandline(
    r"C:\clustalo.exe", infile=in_file,
    distmat_full=True, percentid=True,
    distmat_out="output.distmat",
    guidetree_out="filtree.dnd", verbose=True,
    outfmt="fasta", outfile=out_file,
    outputorder="tree-order",
    seqtype="rna",
    threads=12,
    force=True)
stdout, stderr = clustalomega_cline()
```

Листинг 5.5: Позивање програма Clustal Omega

Резултат поравнања налази се у датотеци `aligned.fasta`, док се матрица сличности (због активиране опције `percentid` у матрици се неће налазити растојања него проценат сличности између поравнатих секвенци) налази у датотеци `output.distmat`.

5.5.2 Филогенетска анализа и конструкција филогенетског стабла

Датотека `aligned.fasta` представља улазну датотеку програма `FastTree`, који ће генерисати филогенетско стабло у `Newick` формату. Овај програм налази се у истом директоријуму као и главни програм који је покренут. Наредбе за покретање програма `FastTree` приказане су у листингу 5.6.

```
output_tree = "stablo.nwk"
subprocess.run(["FastTree", "-nt", "aligned.fasta"],
               check=True, stdout=open(output_tree, "w"))
```

Листинг 5.6: Позивање програма `FastTree`

Програм `FastTree` покренут је позивањем функције `subprocess` која је део стандардне библиотеке програмског језика Пајтон. Опција `nt` означава да се квенце чине нуклеотиди, а не аминокиселине што је случај са протеинским секвенцама. Опција `check` омогућава да се све грешке у раду програма `FastTree` прикажу и у Пајтон програму. Помоћу последње опције `stdout` поставља се референца ка излазној датотеци што је датотека `stablo.nwk`.

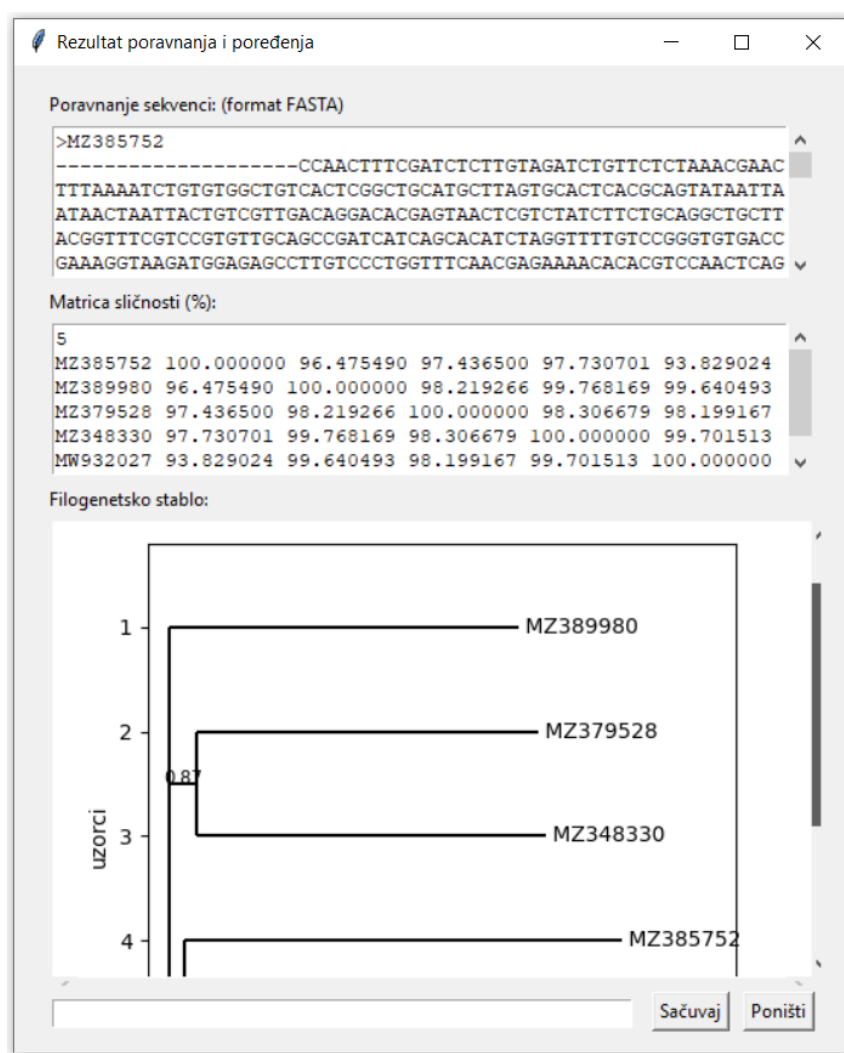
Цртање филогенетског стабла реализовано је помоћу функција из библиотеке `Biopython` и `Matplotlib`. Најпре се помоћу функције `read` чита садржај датотеке `stablo.nwk`, а потом се прочитано филогенетско стабло црта помоћу функције `draw`. На крају се, помоћу функције `savefig` из библиотеке `Matplotlib`, слика филогенетског стабла чува у датотеци `filo.png`. Искључивањем опције `do_show` спречава се да се стабло одмах након цртања прикаже у интерактивном прозору. Ово стабло ће бити касније приказано уз остале резултате поређења у посебно генерисаном прозору. Сви претходно описани поступци приказани су у листингу 5.7.

```
tree = Phylo.read(output_tree, "newick")
fig = plt.figure(figsize=(5,5), dpi=100)
axes = fig.add_subplot(1,1,1)
Phylo.draw(tree, axes=axes, do_show=False)
axes.set_xlabel("evolutivna udaljenost")
axes.set_ylabel("uzorci")
plt.savefig("filo.png")
```

Листинг 5.7: Читање, приказ и чување филогенетског стабла

5.5.3 Приказ и чување резултата поравнања

Резултати поравнања нуклеотидних секвенци изабраних узорака приказују се у додатном прозору. У овом прозору приказано је поравнање секвенци у FASTA формату, матрица сличности и филогенетско стабло. На крају, у доњем делу, се налази и текстуално поље за унос путање и дугме за чување резултата поравнања. Прозор у коме се приказују резултати поравнања је приказан на слици 5.8.



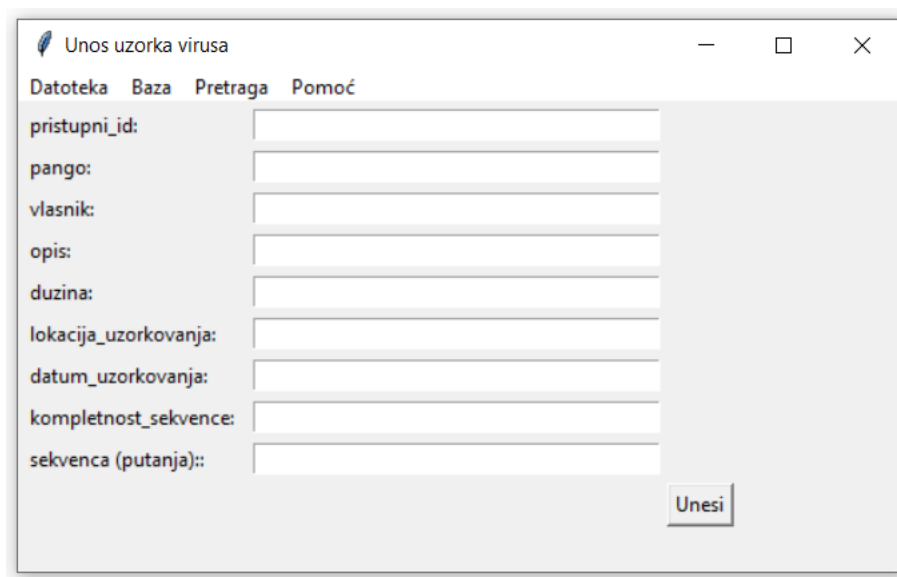
Слика 5.8: Резултат поравнања нуклеотидних секвенци узорака

У менију у картици **Datoteka** постоји опција за отварање и приказ садржаја датотеке са резултатима поређења. Могуће је отварање и више резултата поређења што може олакшати упоредну анализу резултата.

Уколико је на почетној страници унет истакнути узорак са којим треба упоредити преостале узорке, у додатном помоћном прозору ће се приказати проценат сличности тог узорака са преосталим узорцима. Сви подаци о сличности биће прочитани из матрице растојања.

5.5.4 Унос и ажурирање података

Апликација садржи странице за унос и ажурирање података о сојевима и узорцима вируса који се чувају у бази података. Због њихове сличне структуре, у наставку ће бити описана само страница за унос новог узорака, док се остале странице могу детаљније прегледати у документацији рада. Приступ страницима за управљање подацима у бази података остварује се путем картице **Baza** у оквиру менија. На слици 5.9 је приказана страница за унос новог узорака.



The screenshot shows a window titled "Unos uzorka virusa" with a menu bar containing "Datoteka", "Baza", "Pretraga", and "Pomoć". The main area contains a form with the following fields: "pristupni_id:", "pango:", "vlasnik:", "opis:", "duzina:", "lokacija_uzorkovanja:", "datum_uzorkovanja:", "kompletnost_sekvence:", and "sekvenca (putanja):". Each field has a corresponding text input box. A "Unesi" button is located at the bottom right of the form.

Слика 5.9: Страница за унос новог узорака вируса

Уколико текстуално поље неког атрибута остане празно, у базу података ће се на одговарајуће место уписати вредност NULL. Текстуална поља атрибута `pristupni_id`, `pango` и `sekvenca` морају имати унете вредности како би се нови узорак унео у базу података. У поље које одговара секвенци уноси се путања ка FASTA датотеци у којој се налази нуклеотидна секвенца узорака.

Кликом на дугме `Unesi` позива се метод `unesiSoj` из класе `Applikacija`.

У листингу 5.8 су приказане наредбе којима се отвара FASTA датотека чија се путања налази у променљивој `vrednost`, а потом се издваја, енкодира и компресује нуклеотидна секвенца узорка.

```
ulaz = open(vrednost, "r")
seq = ulaz.read()
seq = seq[seq.find('\n'):]
seq = seq.replace('\n', '')
seq1 = seq.encode()
kompresovano = zlib.compress(seq1)
```

Листинг 5.8: Читање и обрада нуклеотидне секвенце из FASTA датотеке

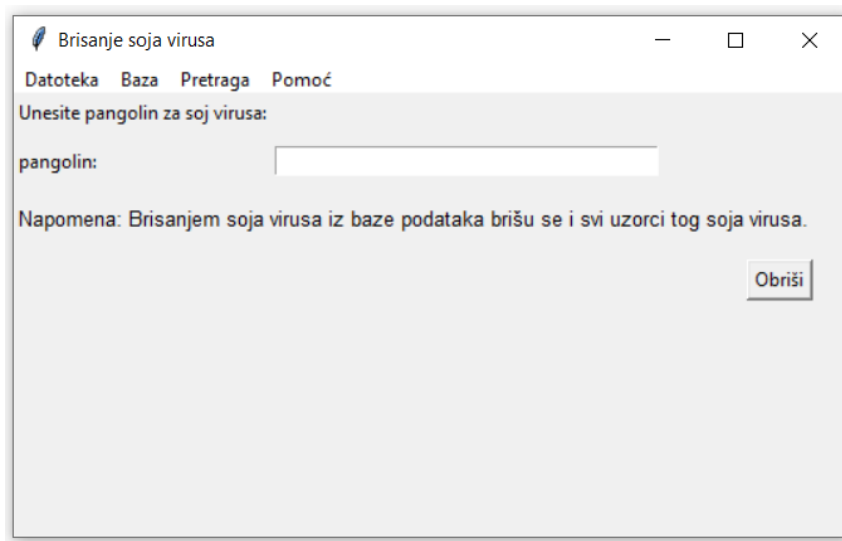
Референце ка свим текстуалним пољима странице додате су у листу `f7entry`. На тај начин је омогућено читање унетих података из функције `unesiSoj`. Након читања извршена је провера исправности унетих података и њихово смештање у листу `vrednosti`. Ако неки податак није унет (и није обавезан) у листу вредности се на одговарајуће место додаје вредност `NULL`. Листинг 5.9 садржи део програмског кода којим се припрема SQL наредба и торка са вредностима за унос узорка у базу података. Након припреме SQL наредба се извршава.

```
sql = "INSERT INTO uzorak (pristupni_id, pango,
                           vlasnik, opis, duzina,
                           lokacija_uzorkovanja,
                           datum_uzorkovanja,
                           kompletnost_sekvenca,
                           sekvenca) VALUES
                           (%s, %s, %s, %s, %s,
                           %s, %s, %s, %s)"
val = tuple(v.strip() for v in vrednosti[:8]) +
      (kompresovano,)
baza = povezivanje_sa_bazom("localhost", "root",
                             "", "baza")
kursor = baza.cursor()
kursor.execute(sql, val)
baza.commit()
```

Листинг 5.9: Унос узорка у базу података

5.5.5 Брисање података

У оквиру картице **Baza** у менију су, поред страница за унос и ажурирање података о сојевима вируса и њиховим узорцима, доступне и странице за њихово брисање из базе података. У овом поглављу рада биће описано брисање соја вируса из базе података. Страница за брисање соја вируса из базе података приказана је на слици 5.10.



Слика 5.10: Страница за брисање соја вируса

Један сој вируса може да има више узорака у бази података. Како је примарни кључ табеле `soj_virusa` уједно и страни кључ у табели `uzorak`, брисањем соја вируса који има бар један узорак у бази података нарушава се услов динамичког референцијалног интегритета дефинисаног у поглављу 2.4.

Приликом дефинисања табела у бази података није омогућено каскадно брисање, па је прво потребно обрисати све узорке одговарајућег соја вируса из базе података, а потом и сам сој вируса, што је и приказано у листингу 5.10.

```
baza = povezivanje_sa_bazom("localhost", "root",  
                             "", "baza")  
  
kursor = baza.cursor()  
sql = "DELETE FROM uзорак WHERE pango = %s"  
kursor.execute(sql, (pango,))  
sql = "DELETE FROM soj_virusa WHERE pango = %s"  
kursor.execute(sql, (pango,))  
baza.commit()
```

Листинг 5.10: Брисање узорака, а потом и соја вируса из базе података

5.5.6 Претрага података

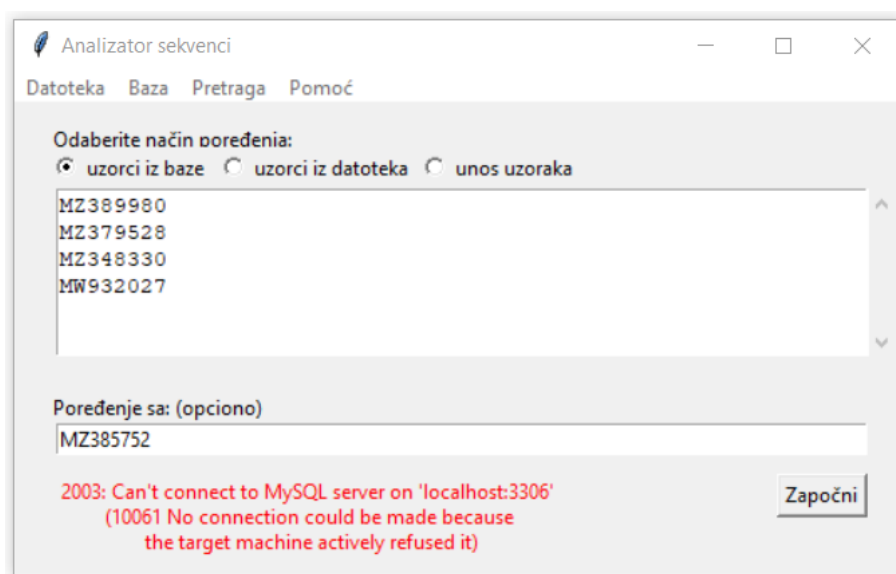
У оквиру картице **Pretraga** доступне су опције за претраживање података у бази. Омогућено је претраживање сојева вируса и узорака, као и коришћење опције напредне претраге, која пружа могућност уноса и извршавања SQL упита. У овом одељку биће представљен поступак претраге сојева вируса у оквиру базе података. На слици 5.11 је приказана страница за претрагу сојева вируса.

Слика 5.11: Страница за претрагу сојева вируса

Уколико ни једно поље није попуњено приказују се сви редови из табеле `soj_virusa`. Свако поље које желимо да прикажемо у резултату мора се означити уносом карактера `*`, док ће се уносом другачије вредности извршити претрага према унетој вредности. Резултати упита приказују се у додатном прозору као што је приказано на слици 5.5.

5.6 Обрада грешака

У раду са базом података и графичким корисничким интерфејсом посвећена је посебна пажња обради потенцијалних грешака, које могу настати током извршавања апликације. Обухваћене су грешке приликом успостављања везе са базом података, неправилног уноса података приликом поравнања и обраде секвенци, грешке изазване неправилно наведеним путањима до конфигурационих или улазних датотека, као и различите друге SQL грешке настале услед неконзистентности у типовима података или нарушавања ограничења дефинисаних у бази података. Све ове грешке су обрађене тако да се апликација не прекида нагло, већ се у одговарајућим лабелама приказује јасна информација о природи проблема, а стање апликације и базе остаје конзистентно, као што је и приказано на слици 5.12.



Слика 5.12: Грешка приликом повезивања са базом података

У табели 5.3 су приказани неки од најзначајнијих изузетака који су обрађени у оквиру апликације.

изузетак	опис грешке
<code>FileNotFoundException</code>	Изузетак који настаје услед покушаја приступа датотеци која не постоји или лоше унетој путањи.
<code>CalledProcessError</code>	Изузетак који се јавља када се спољашњи процес покренут позивом функције <code>subprocess</code> заврши са кодом грешке.
<code>ValueError</code>	Настаје када функција добије аргумент исправног типа, али неодговарајуће вредности.
<code>mysql.connector.Error</code>	Изузеци који се јављају приликом комуникације са MySQL базом података, укључујући синтаксне и грешке приликом повезивања.
<code>Exception</code>	Општи изузетак који представља класу за све уобичајене грешке.

Табела 5.3: Преглед најзначајнијих изузетака обрађених у раду

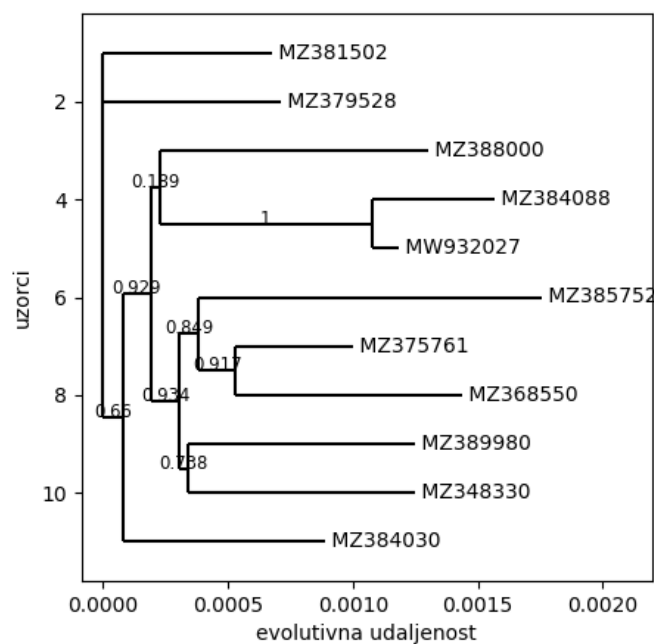
5.7 Приказ и анализа добијених резултата

У овом поглављу представљени су резултати добијени поравнањем свих једанаест нуклеотидних секвенци које се налазе у бази података. Као што је већ речено у питању су узорци различитих сојева вируса SARS-CoV-2. Како би се анализирао филогенетска повезаност и еволутивна разноврсност различитих сојева вируса унутар једне географске популације, одабрани узорци потичу из Сједињених Америчких Држава. Као резултат вишеструког поравнања добијен покретањем алата **Clustal Omega**, генерисана је датотека са поравнањем у FASTA формату, а потом је позивањем програма **FastTree**, на основу добијеног поравнања генерисано филогенетско стабло приказано на слици 5.13. Добијено стабло садржи више унутрашњих чворова са различитим нивоима подршке.

Две међусобно најсличније секвенце, са веома малом еволутивном разликом од 0,000033820 су секвенце MZ348330 (сој P.1) и MZ389980 (сој B.1.1.7) које чине један кластер. Подршка за овај чвор износи 0,738, што указује на умерен ниво поверења у њихову сродност.

Други кластер чине секвенце MZ375761 (сој P.2) и MZ368550 (сој P.3) за које је еволутивна разлика 0,000151186 са подршком 0,917 што указује да су ова два узорка могла бити из истог епидемиолошког таласа и може да представља дивергенцију унутар једног локалног жаришта.

Заједно са узорком MZ385752 (сој C.37) ови узорци чине један разгранати кластер са умереном подршком чвора од 0,849.



Слика 5.13: Филогенетско стабло за поравнање нуклеотидних секвенци

Још један кластер са највећом могућом подршком 1 и еволутивном разликом 0,000847220 чине нуклеотидне секвенце узорака MZ384088 (сој B.1.617.1) и MW932027 (сој B.1.617.2). Ово означава да су ова два соја највероватније настали један од другог или да деле заједничког претка. Овај пар узорака заједно са секвенцом MZ388000 (сој B.1.525) уз слабу подршку од 0,189 чини шири кластер са могућим заједничким пореклом.

У добијеном филогенетском стаблу може се уочити да узорци чије су ознаке MZ385752, MZ375761, MZ368550, MZ389980, MZ348330 чине један већи монофилетски кластер са унутрашњим чворовима са високом подршком.

Узорци са ознакама MZ381502 (сој B.1.526) и MZ379528 (сој B.1.351) гранају се ближе корену и имају већу еволутивну дистанцу у односу на све остале узорке што може да указује на њихову ранију дивергенцију или већу генетску варијацију у односу друге узорке.

Глава 6

Закључак

У овом раду је развијена апликација у програмском језику Пајтон која омогућава рад са базом података у којој се чувају биолошки подаци. Апликација подржава одабир и поравнање нуклеотидних секвенци одабраног скупа узорака, унос, ажурирање, брисање и претрагу сојева вируса и њихових узорака у бази података. Нуклеотидне секвенце су поравнате коришћењем програма *Clustal Omega*, док је филогенетско стабло конструисано помоћу програма *FastTree*.

Добијени резултати поравнања једанаест узорака вируса SARS-CoV-2 указују да се у скупу узорака може уочити један истакнути већи монофилетски кластер коме припадају нуклеотидне секвенце узорака чији су сојеви означени са Alpha, Gamma, Zeta, Theta и Lambda. Припадност ових сојева истом кластеру упућује на постојање сличних мутација, њихово заједничко порекло или сличан правац њиховог еволутивног развоја.

Сви иницијално постављени циљеви овог рада су успешно реализовани, али и даље постоји простор за унапређење и даљи развој базе података, корисничког интерфејса апликације и нових функционалности. Унапређење корисничког интерфејса може се остварити кроз развој бољег визуелног приказа података у табелама, додавање филтера и могућности за сортирање резултата. Развој додатних функционалности укључује: извоз и генерисање извештаја у формате CSV и PDF, чување историје претходних SQL упита, увођење провере идентитета корисника и пријаве, као и развој подршке за истовремени рад више корисника над базом података.

Литература

- [1] S. Esakkirajan S. Sumathi. *Fundamentals of Relational Database Management Systems*. Springer, 2007.
- [2] Paul N. Weinberg James R. Groff. *The Complete Reference to SQL*. Osborne/-McGraw-Hill, 1999.
- [3] Wesley Chan. *Core Python*. Prentice Hall PTR, 2001.
- [4] *IEEE Spectrum's Top Programming Languages 2022*. <https://spectrum.ieee.org/top-programming-languages-2022/>. Приступљено: 10. јул 2023.
- [5] Vladimir Blagojevic. *Relacione baze podataka I*. ICNT, 2006. ISBN: 86-86531-07-5.
- [6] Charles Bachman. „The origin of the integrated data store (IDS): The first direct-access DBMS”. У: *IEEE Annals of the History of Computing* 31 (Окт. 2009), стр. 42–54. DOI: 10.1109/MAHC.2009.110.
- [7] Donald D. Chamberlin и Raymond F. Boyce. *SEQUEL: A Structured English Query Language*. SIGFIDET '74. Ann Arbor, Michigan: Association for Computing Machinery, 1974, стр. 249–264. ISBN: 9781450374156. DOI: 10.1145/800296.811515. URL: <https://doi.org/10.1145/800296.811515>.
- [8] Donald D. Chamberlin. „Early History of SQL”. У: *IEEE Annals of the History of Computing* 34.4 (2012), стр. 78–82. DOI: 10.1109/MAHC.2012.61.
- [9] E. F. Codd. „A Relational Model of Data for Large Shared Data Banks”. У: *Commun. ACM* 13.6 (1970), стр. 377–387. ISSN: 0001-0782. DOI: 10.1145/362384.362685. URL: <https://doi.org/10.1145/362384.362685>.
- [10] Doug Stacey. „Replication: DB2, Oracle, or Sybase?” У: *SIGMOD Rec.* 24.4 (Дец. 1995), стр. 95–101. ISSN: 0163-5808. DOI: 10.1145/219713.219774. URL: <https://doi.org/10.1145/219713.219774>.

- [11] Kristi Berg, Dr. Tom Seymour и Richa Goel. „History Of Databases”. У: *International Journal of Management & Information Systems (IJMIS)* 17 (Дец. 2012), стр. 29. DOI: 10.19030/ijmis.v17i1.7587.
- [12] Hibatullah Alzahrani. „Evolution of Object-Oriented Database Systems”. У: *Global Journal of Computer Science and Technology* 16.C3 (Мај 2016), стр. 37–40.
- [13] Jerzy Letkowski. „Doing database design with MySQL”. У: *Journal of Technology Research* Volume 6 (Јан. 2015).
- [14] Kevin P. Gaffney и др. „SQLite: Past, Present, and Future”. У: *Proc. VLDB Endow.* 15 (2022), стр. 3535–3547.
- [15] Michael Witham, Isaiah Bender и Rahul Gomes. „Comparative Analysis of MariaDB’s Performance Efficiency as a Suitable Replacement for MySQL”. У: Апр. 2019.
- [16] C.J. Date и H. Darwen. *Databases, Types and the Relational Model: The Third Manifesto*. Addison-Wesley, 2007. ISBN: 9780321399427.
- [17] Jan Paredaens и др. *The Nested Relational Database Model*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, стр. 177–201. ISBN: 978-3-642-69956-6. DOI: 10.1007/978-3-642-69956-6_7. URL: https://doi.org/10.1007/978-3-642-69956-6_7.
- [18] Sebastian Bassi. *Python for bioinformatics (2nd ed.)* Chapman и Hall/CRC, 2017.
- [19] Samuel Mbuguah, Victor Mony и Geoffrey Nyabuto. „Architectural Review of Client-Server Models”. У: *International Journal of Scientific Research and Engineering Trends* 10 (Јан. 2024), стр. 139–143.
- [20] Kay Vogelgesang Kai Seidler. *Das XAMPP-Handbuch, Der offizielle Leitfaden zu Einsatz und Programmierung*. Addison-Wesley, 2006. ISBN: 978-3-8273-2281-4.
- [21] Преузимање програма XAMPP: <https://www.apachefriends.org/download.html>. Приступљено: 20. август 2024.
- [22] Преузимање програма Python IDLE: <https://www.python.org/downloads/release/python-3820/>. Приступљено: 15. јануар 2025.
- [23] Документација за MySQL Connector: <https://dev.mysql.com/doc/connector-python/en/>. Приступљено: 20. јануар 2025.

- [24] Аргументи конекције: <https://dev.mysql.com/doc/connector-python/en/connector-python-connectargs.html>. Приступљено: 21. јануар 2025.
- [25] A. Lukaszewski и A. Reynolds. *MySQL for Python*. Community experience distilled. Packt Pub., 2010. ISBN: 9781849510196.
- [26] Paulien Hogeweg. „The Roots of Bioinformatics in Theoretical Biology”. У: *PLoS computational biology* 7 (Мар. 2011), e1002021. DOI: 10.1371/journal.pcbi.1002021.
- [27] Ardeshir Bayat. „Bioinformatics: Science, medicine, and the future”. У: *BMJ (Clinical research ed.)* 324 (Мај 2002), стр. 1018–22. DOI: 10.1136/bmj.324.7344.1018.
- [28] Robert Molidor и др. „New trends in bioinformatics: From genome sequence to personalized medicine”. У: *Experimental gerontology* 38 (Нов. 2003), стр. 1031–6. DOI: 10.1016/S0531-5565(03)00168-2.
- [29] Alexander Johnson et al. Bruce Alberts Rebecca Heald. *Molecular Biology of the Cell. 7th edition*. W. W. Norton и Company, 2022. ISBN: 9780393884821.
- [30] Kristen Ogden, Sarah McDonald и John Patton. „Mechanism of Intraparticle Synthesis of the Rotavirus Double-stranded RNA Genome”. У: *The Journal of biological chemistry* 285 (Мар. 2010), стр. 18123–8. DOI: 10.1074/jbc.R110.117671.
- [31] Kenneth Rosen и др. *Handbook of Discrete and Combinatorial Mathematics*. Св. 84. Јан. 2010. DOI: 10.2307/3621723.
- [32] Lusheng Wang и Tao Jiang. „On the Complexity of Multiple Sequence Alignment”. У: *Journal of computational biology : a journal of computational molecular cell biology* 1 (Феб. 1994), стр. 337–48. DOI: 10.1089/cmb.1994.1.337.
- [33] Peter Clote и Rolf Backofen. *Computational molecular biology. An introduction*. Wiley, Јан. 2000.
- [34] Fabian Sievers и Desmond Higgins. „Clustal Omega, Accurate Alignment of Very Large Numbers of Sequences”. У: *Methods in molecular biology (Clifton, N.J.)* 1079 (Јан. 2014), стр. 105–16. DOI: 10.1007/978-1-62703-646-7_6.
- [35] Fabian Oredame et al. Sievers. „Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega”. У: *Molecular systems biology* 7 (Окт. 2011), стр. 539. DOI: 10.1038/msb.2011.75.

- [36] Gordon Blackshields и др. „Sequence Embedding for Fast Construction of Guide Trees for Multiple Sequence Alignment”. У: *Algorithms for molecular biology : AMB* 5 (Мај 2010), стр. 21. DOI: 10.1186/1748-7188-5-21.
- [37] Johannes Söding. „Protein homology detection by HMM–HMM comparison”. У: *Bioinformatics* 21.7 (Нов. 2004), стр. 951–960. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bti125.
- [38] Програм *Clustal Omega*: <http://www.clustal.org/omega/>. Приступљено: 1. фебруар 2024.
- [39] Lindell Bromham. *An Introduction to Molecular Evolution and Phylogenetics*. АБГ. 2023. ISBN: 9780198736363. DOI: 10.1093/hesc/9780198736363.001.0001.
- [40] Barry G. Hall. *Phylogenetic Trees Made Easy: A How-To Manual*. 5th. Sunderland, MA / New York, NY: Sinauer Associates (an imprint of Oxford University Press), 2017. ISBN: 9781605357102.
- [41] Manolis Kellis и James Galagan. *Computational Biology: Genomes, Networks, and Evolution*. MIT OpenCourseWare course 6.047 / 6.878. Cambridge, MA: Massachusetts Institute of Technology, 2008. URL: <https://ocw.mit.edu>.
- [42] Kieran Boyce, Fabian Sievers и Desmond Higgins. „Instability in progressive multiple sequence alignment algorithms”. У: *Algorithms for Molecular Biology* 10 (Окт. 2015), стр. 26. DOI: 10.1186/s13015-015-0057-1.
- [43] Gaston Gonnet. „Surprising results on phylogenetic tree building methods based on molecular sequences”. У: *BMC bioinformatics* 13 (Јун 2012), стр. 148. DOI: 10.1186/1471-2105-13-148.
- [44] Програм *FastTree*: <https://morgannprice.github.io/fasttree/>. Приступљено: 5. мај 2025.
- [45] Morgan N. Price, Paramvir S. Dehal и Adam P. Arkin. „FastTree: Computing Large Minimum-Evolution Trees with Profiles Instead of a Distance Matrix”. У: *Molecular Biology and Evolution* 26.7 (2009), стр. 1641–1650. DOI: 10.1093/molbev/msp077.
- [46] Joseph Felsenstein. „Confidence Limits on Phylogenies: An Approach Using the Bootstrap”. У: *Evolution* 39.4 (1985), стр. 783–791. DOI: 10.2307/2408678.

- [47] Morgan Price, Paramvir Dehal и Adam Arkin. „FastTree 2 – Approximately Maximum-Likelihood Trees for Large Alignments”. У: *PloS one* 5 (Мар. 2010), e9490. DOI: 10.1371/journal.pone.0009490.
- [48] Библиотека Tkinter: <https://docs.python.org/3/library/tkinter.html>. Приступљено: 3. март 2025.
- [49] Праћење сојева вируса SARS-CoV-2 од стране Светске здравствене организације: <https://www.who.int/activities/tracking-SARS-CoV-2-variants>. Приступљено: 13. март 2025.
- [50] Gisaid номенклатура: <https://gisaid.org/resources/statements-clarifications/clade-and-lineage-nomenclature-aids-in-genomic-epidemiology-of-active-hcov-19-viruses/>. Приступљено: 25. јануар 2025.
- [51] Rasha Emad Mansour и Iman Naga. „Comparative genotyping of SARS-CoV-2 among Egyptian patients: near-full length genomic sequences versus selected spike and nucleocapsid regions”. У: *Medical Microbiology and Immunology* 212 (Окт. 2023), стр. 1–10. DOI: 10.1007/s00430-023-00783-8.
- [52] Jens Kuhn et al. Frank Konings Mark Perkins. „SARS-CoV-2 Variants of Interest and Concern naming scheme conducive for global discourse”. У: *Nature Microbiology* 6 (Јун 2021), стр. 821–823. DOI: 10.1038/s41564-021-00932-w.
- [53] FASTA формат за нуклеотидне секвенце: <https://www.ncbi.nlm.nih.gov/genbank/fastafORMAT/>. Приступљено: 4. фебруар 2024.

Биографија аутора

Милош Арсић (Београд, Србија, 11. април 1994.) је дипломирани математичар, Математичког факултета Универзитета у Београду. Ради као наставник рачунарства и информатике у Математичкој гимназији у Београду, чији је и бивши ученик.

У Регионалном центру за младе таленте, Београд 1 - Земун ради као сарадник на предмету математика где је и ментор научно-истраживачких радова ученика. Током студија био је практикант компаније РТ-РК где је радио у тиму за програмирање сет-топ бокс (енг. *set-top box*) уређаја.

Један је од дописника колумне *Открића* часописа *Елементи*, Центра за промоцију науке. Аутор је многобројних поставки, мултимедијалних садржаја и обука између којих се истиче поставка *Како су настали бројеви* која је одржана у оквиру представљања Математичког факултета на већем броју фестивала у Србији.

Добитник је прве награде на такмичењу Matf2016++ за најбољи истраживачки рад из области рачунарства и информатике. Поред тога добитник је и признања Математичког факултета за показану изузетну хуманост током студија. Аутор је и два рада објављена редом у зборнику Задужбине Андрејевић и часопису Спрингер. У слободно време бави се волонтерским и хуманитарним радом, иконописом и планинарењем.