УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Милица Д. Гњатовић

ИМПЛЕМЕНТАЦИЈА СИСТЕМА ЗА ОЦЕЊИВАЊЕ ЗАДАТАКА ИЗ SQL-А

мастер рад

Београд, 2024.

Ментор:

др Ивана Танасијевић, доцент
Универзитет у Београду, Математички факултет

Чланови комисије:

проф. др Саша Малков, ванредни професор
Универзитет у Београду, Математички факултет

др Данијела Симић, доцент
Универзитет у Београду, Математички факултет

Датум одбране:

Студентима који су инспирисали овај рад

Наслов мастер рада: Имплементација система за оцењивање задатака из SQL-а

Резиме:

Проблем аутоматизације оцењивања радова је занимљив са информатичке стране јер повезује аутоматско одлучивање са ширим системом који има за циљ да паралелно у реалном времену опслужује већи број корисника.

У овом раду је изложена имплементација система за оцењивање задатака из SQL-а. Систем је организован микросервисно, састоји се из сервера, клијента и апликације задужене за оцењивање. Систем је имплементиран на начин да је релативно једноставно додати нове оцењиваче. Серверски део и оцењивач су имплементирани у језику *Java* уз коришћење додатних библиотека, док је клијентска страна имплементирана у *Angular*-у.

Кључне речи: информациони систем, базе података, имеплементација, оцењивач

Садржај

Глава 1

Увод

Обавезан део учења програмирања је писање кода који испуњава захтеве задатака којим се проширује и утврђује знање. Након што студент уради задатак главно питање је да ли је решење тачно. Постоји неколико приступа за разматрање.

Уколико је дато тачно решење задатка онда студент може релативно једноставно да провери да ли је његово решење тачно. Проблем са овим приступом је то што студенти могу превише да се ослоне на тачно решење, а да не размишљају о другим начинима да се задатак реши. Велики број задатака може да се реши на више начина. За студенте је корисно да уче да размишљају тиме што ће покушати да реше задатак на различите начине, што није могуће уколико они прво погледају решење.

Уколико студент нема тачно решење задатка може бити тешко проценити да ли је његово решење тачно. Студенти имају могућност да консултују асистенте, али аутоматизација даје бржи одговор.

Идеално би било да студент има апликацију која би проверила да ли је његово решење тачно. Циљ овог рада је креирање једне управо такве апликације. У овом раду ће фокус бити на задацима из *SQL-a* који захтевају издвајање одговарајућих података из базе, а без измене структуре базе и података који се налазе у њој. Систем је имплементиран на начин да је релативно једноставно проширити га да прегледа задатке из других програмских језика.

Аутоматизација представља једну од занимљивих и популарних области информатике. Област је значајна јер има широку примену. Циљ аутоматизације је брже завршити мануални посао, у те сврхе се често користе разне

скрипте.

Много послови који су се некад радили сатима се данас завршавају у неколико кликова, попут аутоматске анализе кода, континуалне интеграције и испоруке кода, као и аутоматског тестирања.

У контексту оцењивања и провере тачности кодова постоји доста техничких изазова. Јако важан елемент је заштита система од корисника, јер корисници оваквих система пишу кодове који могу бити произвољно опасни по систем, а морају се извршити да би били оцењени.

Додатни изазов када је систем за оцењивање у питању је провера исправности система. Не смеју постојати лажно тачни, али ни лажно нетачни резултати оцењивања. Важно је имати релевантне примере за тестирање, при чему је тешко предвидети шта све корисници могу да осмисле.

Када се праве системи за велики број корисника мора се водити рачуна о паралелизацији, скалабилности и ефикасности таквих система и имати у виду колико ће системског времена заузети сваки корисник. Када је оцењивање радова у питању важно је имати у виду да корисници могу да пишу неефикасне кодове, а да се целокупан систем може успорити уколико се оцењује велики број радова одједном. Јасно је да сваки рад мора имати ограничено време за оцењивање. Одређивање те временске границе може бити захтевно јер треба имати у виду да систем не врши само оцењивање, већ има и додатне активности које заузимају ресурсе, попут одржавања корисничких налога и доступних задатака.

Када је реч о великим платформама за оцењивање оне су већином имплементиране коришћењем императивних програмских језика. Могу се разматрати начини на који један код извршава и проверава други код. Када је у питању оцењивања задатака из *SQL* потребно је дохватити податке из базе извршавањем корисничког упита, а затим проверити да ли су добијени подаци који се очекују. Многи императивни програмски језици користе библиотеке којима се дохватају подаци из базе. Програм мора имати информације о бази са којом ради на овај начин, а често је потребан значајан део кода који описује структуру базе. Овакав оцењивач би морао значајно да се измени да би користио другу базу. Са друге стране, неки програмски језици омогућавају извршавање *shell* команди, што омогућава директно извршавање упита без тога да је програм оцењивача свестан структуре базе. У овом раду ће бити представљен оцењивач који користи овај приступ и који се лако прилагођава

другим базама података.

У овом раду ће бити описана архитектура и имплементација система за оцењивање задатака из *SQL-a*, као и његова ограничења.

Глава 2

Оцењивачи

Оцењивачи који проверавају тачност различитих врсти задатака из програмирања већ постоје, па ће овде бити размотрени неки од њих.

Једна од главних инспирација за овај рад је била платформа *DrWebGrader*. Ова платформа је креирана као као мастер рад под насловом *Оцењивач задатака из програмирања - имплементација сложеног софтвера у модерном C++-у* [19]. Овај оцењивач је првенствено намењен задацима предмета *Програмирање 1*, *Програмирање 2*, као и *Оперативни системи*, који користе језике *C* и *C++*.

Претходно поменути оцењивач је имплементиран у језику *C++* и користи систем библиотека. Да би се оценили, на пример, задаци из програмског језика *Python* или програмског језика *Java* потребно је да се имплементира одговарајућа библиотека и дода серверу.

Како *DrWebGrader* првенствено ради са императивним програмским језицима, он поседује одговарајући механизам заштите система од његових корисника. Корисник у *C-у* може да напише програм који ће обрисати неки важан документ оцењивача. Оцењивач ће тај потенцијално опасан код извршити у процесу оцењивања, што може угрозити функционисање система. Заштита је јако важна компонента оваквих система. У овом раду је описан једна оцењивач задатака и начин на који је он заштићен од својих корисника.

Још једна платформа са могућношћу оцењивање задатака из програмирања је *Петља* [11]. Ова платформа је првенствено намењена ученицима основних и средњих школа за учење различитих области програмирања. Платформа има лекције, квизове, као и практичне задатке. Практични задаци дају могућност ученицима да раде задатке, док платформа даје одговор

да ли је решење тачно.

Приликом интервјуисања кандидата за програмерске позиције многе компаније користе различите платформе за проверу знања. То су платформе попут *Codility*, *HackerRank*, *HireVu* и *CodeSignal*. Најчешће кандидати добију текст задатка и део кода који треба да допуне својом имплементацијом. Кодирање се ради у веб прегледачу. Ове платформе су одличне зато што извршавају код и кандидат не мора да има инсталиран програм. Овим се такође избегавају проблеми са верзијама програма и библиотека које кандидат има локално инсталиране. Неке од платформи имају могућност да изврше код за неколико тест улаза чиме кандидат одмах добија одговор колико успешно његово решење ради. Оно што је код ових платформи добро је што корисник може да изврши код над својим улазним параметрима, без извршавања тест примера. Друга добра ствар је што су улазне вредности за тест примере сакривене од корисника, чиме се смањује могућност да корисник намести резултат. Још једна компонента ових оцењивача је да компаније веома често кодове кандидата извршавају над другим скупом тест примера чиме се још једном проверава исправност њихових решења.

Глава 3

Технологије

Од настанка рачунара до данас је креиран велики број програмских језика и алата који се могу користити при развоју апликација. Овде су представљене технологије које ће бити коришћене у имплементацији система који је предмет овог рада.

3.1 Систем за управљање базом података *Db2*

База података представља скуп података који се чувају на рачунару. Систем за управљање базама података (СУБП) је софтвер који омогућава корисницима да приступе подацима који су сачувани у бази података. Постоји више врсти база података, на пример, релационе, нерелационе, објектне, документне. Овде ће фокус бити на релационим базама података.

Релационе базе података је осмислио Едгар Код (*Edgar Codd*). Он је као истраживач у *IBM* лабораторији 1970. године објавио рад који даје идеју о организацији података у табеле са одређеним правилима [14]. За табеле релационих база података важи да сваки ред има примарни кључ који га јединствено одређује. Сви редови табеле се међусобно разликују бар по једној вредности.

СУБП *Db2* је креиран од стране компаније *IBM*. У протеклих 40 година систем је знатно унапређен и данас представља један од често коришћених СУБП. О квалитету овог система сведочи чињеница да га користе компаније попут *American Express*, *JPMorgan Chase* и *Walmart* [17].

СУБП *Db2* пружа добре перформансе и скалабилност. Систем поседује напредне механизме оптимизације који му омогућавају да ефикасно обрађује

велике количине података. Неки од механизама оптимизације овог система су материјализовани упити и индекси. Материјализовани упити омогућавају извршавање комплексног упита по потреби и чување његовог резултата, док се подацима који се налазе у упиту може приступати брзо и без додатних израчунавања. Креирањем индекса се омогућава бржи приступ подацима великих табела. Брзини извршавања доприноси и паралелно процесирање. Претходно наведени елементи су само неки од механизама које овај систем нуди корисницима у циљу повећања перформанси у раду са базом података.

Када се ради са великом количином података треба имати у виду њену заштиту. Систем *Db2* нуди могућност креирања погледа којим би се онемогућио приступ свим редовима и колонама од стране неауторизованих корисника. Поред тога, постоји могућност давања права приступа табелама, погледима и схемама одговарајућим корисницима и групама корисника. Може се дефинисати дозвољени ниво приступа, односно да ли корисник може само да гледа податке или може и да их мења. Одређени корисници могу имати могућност мењања података, али не и мењања структуре базе.

Систем *Db2* је доступан на *Windows* и *UNIX* системима. Поред тога, *IBM* је креирао *Db2 docker* слику [3]. Овим је омогућено коришћење овог СУБП без његовог инсталирања директно на рачунару, што знатно може олакшати подешавање система. Ова слика је намењена за развојне сврхе, не и продукцијско окружење, али ће за потребе овог рада бити довољна.

Овај систем има добру подршку и документацију. Како систем постоји већ дуги низ година, поред документације су креирани и разноврсни туторијали и заједнице људи који користе управо овај систем.

Овде се може размотрити коришћење другог СУБП, попут *MySQL*. Предност *Db2* у односу на овај систем се огледа у вертикалној скалабилности [10]. Односно, повећање хардверских ресурса, попут процесорске моћи, ће значајније допринети перформансама система *Db2*.

3.2 Програмски језик *Java* и његове библиотеке

Програмски језик *Java* је 1995. године објавила компанија *Sun Microsystems*, док је 2010. године одржавање језика преузела компанија *Oracle* [18]. *Java* је иницијално замишљена као програмски језик за дигиталне уређаје, али је

прешла дуг пут од те иницијалне идеје. *TIOBE* индекс који мери популарност програмског језика на основу интернет претрага тренутно ставља *Javu* на четврту позицију [12]. У последњих 20 година *Java* је веома често заузимала прву позицију и није се спуштала испод четврте.

Java је објектно оријентисан језик и све у њој је базирано на објектима. Основни концепти објектно оријентисаног програмирања укључују наслеђивање, полиморфизам и енкапсулацију. *Java* подржава једноструко наслеђивање, док једна класа може да имплементира више интерфејса.

Java је независна од платформе и њене апликације могу да се покрећу на великом броју оперативних система, укључујући *Windows*, *Linux* и *macOS*. За разлику од *C*-а и *C++*-а, *Java* кодови се компајлирају до бајткода који даље интерпретира Јава виртуелна машина (енг. *Java Virtual Machine*). Бајткод може да се дистрибуира и покреће на било ком оперативном систему.

Java је робустан програмски језик. Поседује сакупљаче отпадака (енг. *garbage collector*) што програмеру олакшава управљање меморијом. Овај језик не подржава показиваче што доприноси сигурности система.

Постоји велики број библиотека написаних за овај програмски језик, укључујући *Hibernate* и *Grizzly*, као и велики број помоћних алата попут *Maven*-а. Они ће бити описани у наставку.

3.2.1 Библиотека *Hibernate*

Hibernate је библиотека која омогућава управљање базама података из *Java* апликације [6]. Ова библиотека омогућава објектно релационо мапирање, односно табеле базе података се пресликавају у класе програма. Омогућено је дохватање података из базе, уношење, измена као и брисање података.

Ова библиотека нуди висок ниво перформанси и скалабилност. Постоји неколико стратегија за дохватање података из базе, од којих су неке ефикасније од других. На пример, нека постоје табела корисника и табела задатака, при чему један корисник може бити повезан са више задатака. Једноставним дохватањем корисника из *Java* апликације се, у зависности од стратегије, дохватају или не дохватају његови задаци. Уколико је стратегија дохватања задатака *eager*, задаци ће бити дохваћени сваки пут, док уколико је стратегија *lazy*, задаци ће бити дохваћени само уколико им се приступа током сесије. Постоји могућност рада са трансакцијама што додатно доприноси флексибилности система.

Ова библиотека је компатибилна са великим бројем СУБП, укључујући *Db2*, *MySQL* и *PostgreSQL*.

Овде се може размотрити и коришћење *JDBC* алата за приступ подацима базе података из *Java* апликације [7]. Коришћењем овог алата се пишу *SQL* упити директно у код, док их *Java* шаље на извршавање. Иако *JDBC* генерално даје боље перформансе, у овом раду се користи *Hibernate* јер омогућава лакше управљање подацима који су моделовани на објектно оријентисан начин и има стратегије дохватања података које знатно поједностављују рад са подацима.

3.2.2 Библиотека *Grizzly*

Grizzly је библиотека отвореног кода намењена развоју серверских апликација [16]. Ова библиотека омогућава асинхроно обрађивање *HTTP*, *WebSocket*, *SSE* и других захтева, што је чини флексибилним алатом.

Коришћењем ове библиотеке је могуће паралелно обрађивати велики број захтева. Библиотека даје могућност конфигурисања сервера, попут одређивања максималног броја захтева који могу да се обраде паралелно и броја захтева који се могу сместити у ред за извршавање.

3.2.3 Алат *Maven*

Maven је алат за управљање *Java* пројектима [9]. Он води рачуна о библиотекама и њиховим међусобним зависностима.

Овај алат захтева креирање конфигурационог *pom.xml* документа. У њему се налази списак свих библиотека које пројекат користи, док ће алат преузети те библиотеке из централизованог репозиторијума. У документу се дефинише и која верзија библиотеке се користи што смањује могућност проблема са верзијама. Ово олакшава развој и дистрибуцију пројеката јер програмер не мора сам да инсталира различите библиотеке, већ алат води рачуна о свему томе.

Овај алат олакшава дељење апликације јер може да изгради извршни фајл који ће у себи садржати све зависне библиотеке. Рачунар који буде покретао тај извршни фајл не мора да има ни једну инсталирану библиотеку, већ само *Java* окружење.

3.3 Алат *Docker*

Docker је алат за контејнеризацију апликација [4]. Он омогућава покретање апликација у изолованим окружењима која се називају контејнери. Изолованост контејнера се огледа у томе да не користи програме система на ком је покренут. Контејнери заузимају мање меморије од традиционалних виртуелних машина и брже се покрећу.

На једном систему може бити покренуто више апликација у различитим контејнерима. Може се конфигурисати који портови су отворени унутар контејнера и на који начин се може комуницирати преко њих.

Контејнерима се олакшава дељење апликација и комуникацију међу апликацијама. Наиме, један корисник може послати контејнер другом кориснику, тако да други не мора ништа додатно да инсталира на свом рачунару, а да може да покрене све у контејнеру као и први корисник.

Контејнери се праве на основу докер слика. Слика је шаблон за креирање контејнера и садржи инструкције са којим системом и програмима треба направити контејнер. На пример, докер слика садржи инструкције за креирање контејнера који је заснован на одређеној верзији *Linux* система и има одређену верзију *Java* окружења.

Docker Hub је попут библиотеке докер слика. Он садржи преко сто хиљада докер слика које корисници могу да преузимају и користе. Свако са налогом може да објави слике овде док год се придржава смерница платформе. Овде се налазе и званичне докер слике које компаније обезбеђују, попут претходно поменуте *Db2* слике.

Контејнер креиран на основу слике може да се проширује инсталацијама додатних програма. Ако креирамо контејнер на основу произвољне *Linux* слике можемо ући у терминал контејнера и инсталирати, на пример, *Java* окружење. Други начин да се идентична ствар уради је да се користи докер фајл. У том фајлу се наводе команде које се извршавају при креирању контејнера. То су, на пример, инсталирање програма, креирање директоријума, копирање документа са локалног система у контејнер, дефинисање порта који ће бити доступан у контејнеру и слично.

Поред алата *Docker* постоји и алат *Docker Compose* који омогућава изградњу групе контејнера одједном. Овим се олакшава покретање система који се састоји из већег броја сервиса који раде у различитим контејнерима.

Ово се постиже креирањем *compose.yaml* фајла у коме се наводе карактеристике сваког потребног контејнера, при чему се могу користити претходно креирани докер фајлови који су се користили за креирање једне слике.

Додатни алати који могу да се користе уз *Docker* су *Docker Swarm* и *Kubernetes* [5][8]. Иако имају разлике, оба алата имају исту сврху, а то је да олакшају управљање већим бројем контејнера. Они дају могућност праћења рада контејнера и пружају механизме за распоређивање оптерећења између чворова.

3.4 Оперативни систем *Linux*

Linux је креирао Линус Торвалдс 1991. године. *Linux* представља језгро оперативног система (енг. *kernel*) које комуницира са хардвером машине на којој се извршава, а надограђује се да би се изградио оперативни систем. Језгро директно управља хардвером, док преостали део оперативног система представља софтвер који пружа удобан кориснички интерфејс и олакшава коришћење рачунара. *Linux* користи више дистрибуција оперативних система који су отвореног кода. Постоје стотине дистрибуција намењених различитим потребама, неке од најпознатијих су *Ubuntu*, *Lubuntu*, *Fedora* и *Debian*.

Како је изворни код јавно доступан и бесплатан корисници могу да га прилагођавају својим потребама и деле са другим корисницима. Овај оперативни систем се користи на рачунарима, мобилним телефонима и разним другим уређајима.

Овај оперативни систем се користи и за следеће намене:

- Сервере, мрежне уређаје попут рутера и свичева,
- Десктоп рачунаре,
- Системе и уређаје који не захтевају графички интерфејс,
- Инстанце у облаку (енг. *cloud*),
- Многе системе за управљање финансијским трансакцијама,
- Велики број суперрачунара.

Како је *Linux* отвореног кода лако се може прилагодити специфичним потребама и омогућити високе перформансе. Може се прилагодити физичкој машини да би искористио могућности хардвера у потпуности.

Оперативни системи *Linux* дистрибуције се сматрају поузданим и стабилним. Новије верзије су компатибилне са старим, али и добар део дистрибуција је међусобно компатибилан.

За разлику од друга два популарна оперативна система, *Windows* и *macOS*, *Linux* је бесплатан.

3.4.1 *Shell*

Shell представља спону између корисника и оперативног система. Он је уједно и интерпретатор командног језика који омогућава корисницима да комуницирају са оперативним системом преко командне линије.

Shell се најчешће користи у *Linux* оперативним системима. У зависности од дистрибуције оперативног система постоји више типова *shell*-а, на пример, *Bash*, *Zsh* и *Fish*. Сви они имају неке заједничке карактеристике, али и одређене разлике.

Shell се може користити и за писање скрипти. Скрипте омогућавају обједињавање више команди у један документ.

3.5 Радни оквир *Angular*

Angular представља радни оквир (енг. *framework*) за развој веб апликација [1]. Креиран је 2010. године од стране компаније *Google*. Он омогућава креирање брзих, поузданих и скалабилних веб апликација.

Angular користи *TypeScript* као програмски језик [13]. Њега можемо гледати као строжи *JavaScript*, штавише он се преводи управо у *JavaScript*. Предност програмског језика *TypeScript* је строга типизираност и провера синтаксе који смањују шансе за прављење грешака, односно грешке се уочавају лакше него у програмском језику *JavaScript*.

Апликације које су креиране у *Angular*-у су једностраничне (енг. *Single Page Applications*). Приказ на страници се састоји из компоненти. Сваку компоненту чине *html* приказ, функционалност компоненте задата *TypeScript*-ом и стилизовање које може бити задато *CSS*-ом, *SCSS*-ом, *Sass*-ом или *Less*-

ом. Стилизовање се може задати и на нивоу апликације, односно применити исти стил на све компоненте.

Поред компоненти *Angular* нуди и креирање сервиса који се могу убацивати у компоненте. На пример, сервис који води рачуна о улогованом кориснику се убацује у компоненте које приказују корисничке информације.

3.6 Микросервисна архитектура

Микросервисна архитектура представља начин организације софтверских система. Микросервисни систем се састоји од више мањих сервиса. Сервис представља апликацију која самостално обавља неку функционалност. Сервиси међусобно успостављају комуникацију и заједнички обављају већи посао. Како се сервиси праве да би обављали специфичне послове, сваки може да користи различите технологије, базе података, чак и различите програмске језике, што доприноси флексибилности система.

Ова архитектура омогућава лакшу поделу посла у тимовима. Наиме, један сервис може да се развија независно од осталих. Тиме се олакшава и тестирање. Ако се развија сервис А, при чему се зна да ће он комуницирати са сервисом Б, можемо имитирати захтев сервиса Б ка сервису А и тиме проверити да ли А даје очекивани одговор.

Како су сервиси независни, уколико је потребна промена верзија једног сервиса то неће имати утицај на цео систем. Ако би имали монолитну апликацију, промена верзије би утицала и на друге делове система.

У микросервисном систему се може десити да је један сервис оптерећенији од осталих. У том случају је потребно скалирати само тај један сервис. Штавише, у неким случајевима је могуће покренути више инстанци једног сервиса и тиме повећати перформансе целог система.

Иако је предност сервиса што су углавном једноставни и обављају једну улогу, може бити изузетно компликовано ускладити све сервисе у један комплексан систем. Додатан проблем може бити брзина комуникације између сервиса, посебно код дистрибуираних система. Наиме, сервиси комуницирају слањем захтева, *HTTP* или неких других, и то их чини споријим.

Глава 4

Архитектура система

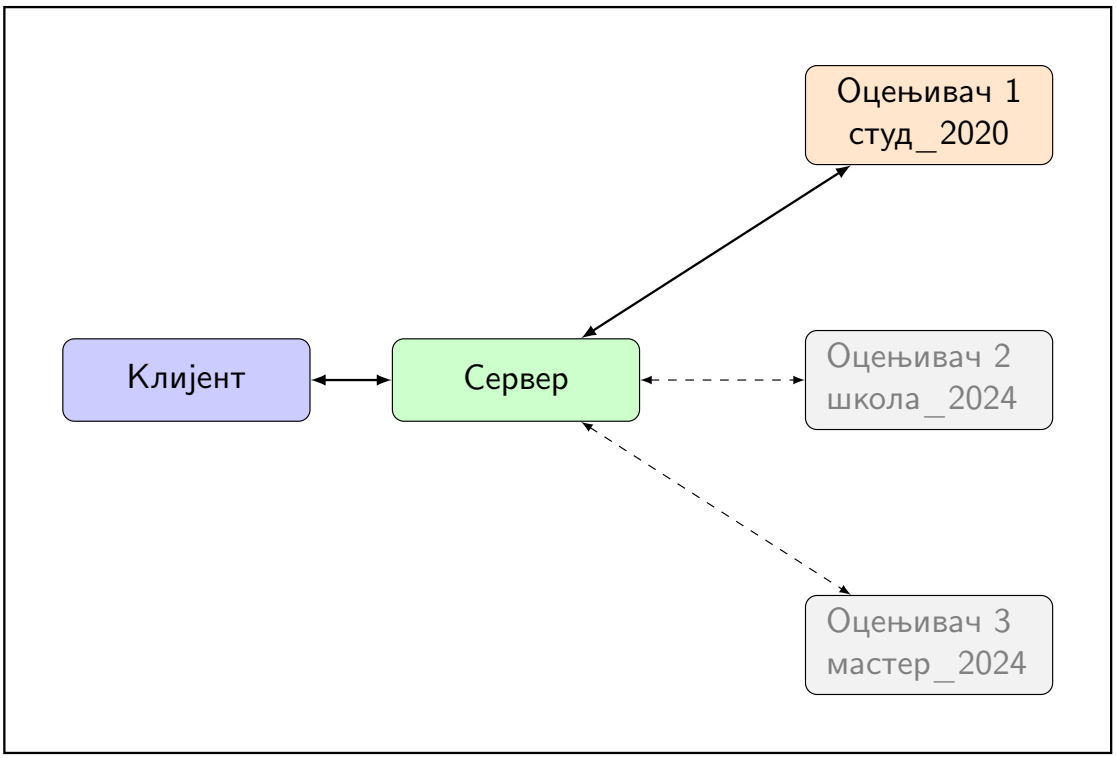
Замислимо да имамо више база података и да задаци подразумевају издвајање података из тих база. Свака база може да користи различит систем за управљање базом података. Уколико би један оцењивач обухватао све ове базе он би добио јако велики број захтева и био би гломазан. Један оцењивач треба да ради са тачно једном базом, односно једним системом за управљање базом података. Овим се добија више мањих оцењивача, који примају само захтеве за задатке за које су они задужени.

Уколико би оцењивач био једна монолитна апликација његово проширивање би захтевало доста рада. Једноставније је имати архитектуру која подразумева компоненте са што вишим нивоом изолованости. Проширивање система се у овом случају своди на додавање нових компоненти и њихово укључивање у већ постојећи систем који ради.

На основу претходног можемо закључити да је најпрактичније да сваки оцењивач буде посебна компонента. Због овакве поставке је изабрана микросервисна архитектура за прављење система. Односно, систем за оцењивање подразумева више сервиса који међусобно комуницирају, али постоје као независне целине, као што је приказано на слици 4.1.

4.1 Опис система

Поред потенцијално више оцењивача, потребан је централни сервис преко ког се врши комуникација. Овај сервис ћемо назвати *Сервер*. Уколико не би имали централизован сервис корисници би морали да контактирају сваки оцењивач да би добили задатке које могу да раде. Сервер чува списак свих



Слика 4.1: Поједностављена архитектура система

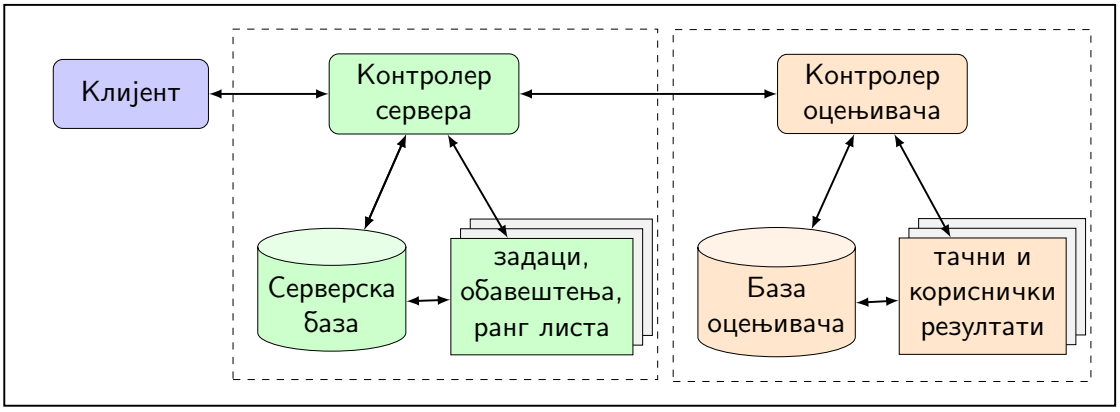
доступних оцењивача, као и задатака који су на њима доступни. Поред тога, сервер води рачуна о корисницима и о томе које могућности система су им доступне.

Да би систем био угоднији за коришћење направљен је додатан клијентски сервис са прилагођеним графичким интерфејсом. Овај сервис ћемо назвати *Клијент*.

Архитектура система са више детаља је приказана на слици 4.2. У овом поглављу је описана логика система, док ће детаљи имплементације бити описани у наредним поглављима. Клијентски сервис се састоји само из клијентске апликације, док серверски и сервис оцењивача имају додатне елементе. Сервер користи базу која садржи задатке, кориснике, табелу доступних оцењивача и још неке табеле које су описане у поглављу 6. Сваки корисник при логовању добија скуп идентичних података, односно доступне задатке, обавештења и ранг листу. Како би се смањио број приступа бази за дохватање ових података постоје додатни, помоћни, документи који их садрже. Ови документи се ажурирају по потреби.

Сервис оцењивача поред главног програма - *Контролера оцењивача*, са-

држи базу над којом се раде задаци и помоћне документе са резултатима. У овом раду се разматра оцењивач који оцењује само задатке у којима се захтева дохватање података из базе, не и њихово мењање. Може се сматрати да клијент који користи овај оцењивач не може да извршава упите који мењају структуру базе или податке. Процес оцењивања би могао да се састоји из извршавања два упита, упита који је дат при задавању задатка и који представља једно тачно решење и корисничког упита, а затим њиховог поређења. Овим приступом би се један исти упит, упит који је дат као тачно решење, извршавао при сваком оцењивању задатка, што је неефикасно. Уместо тога, упит се извршава једном и резултат извршавања се чува у одговарајући помоћни документ оцењивача. Дакле, једну групу помоћних докумената оцењивача чини скуп резултата извршавања тачних упита који су дати као решења задатака. Други скуп помоћних докумената оцењивача чине документи који садрже резултат извршавања упита корисника чији се рад тренутно оцењује. Након што се рад оцени помоћни документ који садржи резултат извршавања корисничког упита се брише, па је овај скуп докумената променљив.



Слика 4.2: Архитектура система

4.1.1 Комуникација између сервиса

У систему се налази један сервер и клијент може да му шаље *HTTP* захтеве за:

- Логовање,
- Креирање новог корисника,

- Оцењивање задатка,
- Постављање питања за задатак.

Када систем користе корисници са већим нивоом привилегија, попут наставника, клијент серверу може да шаље и додатне захтеве, попут захтева за:

- Додавање и измену задатака,
- Одговор на питања других корисника,
- Постављање обавештења.

Сервер оцењивачу може да шаље захтеве за:

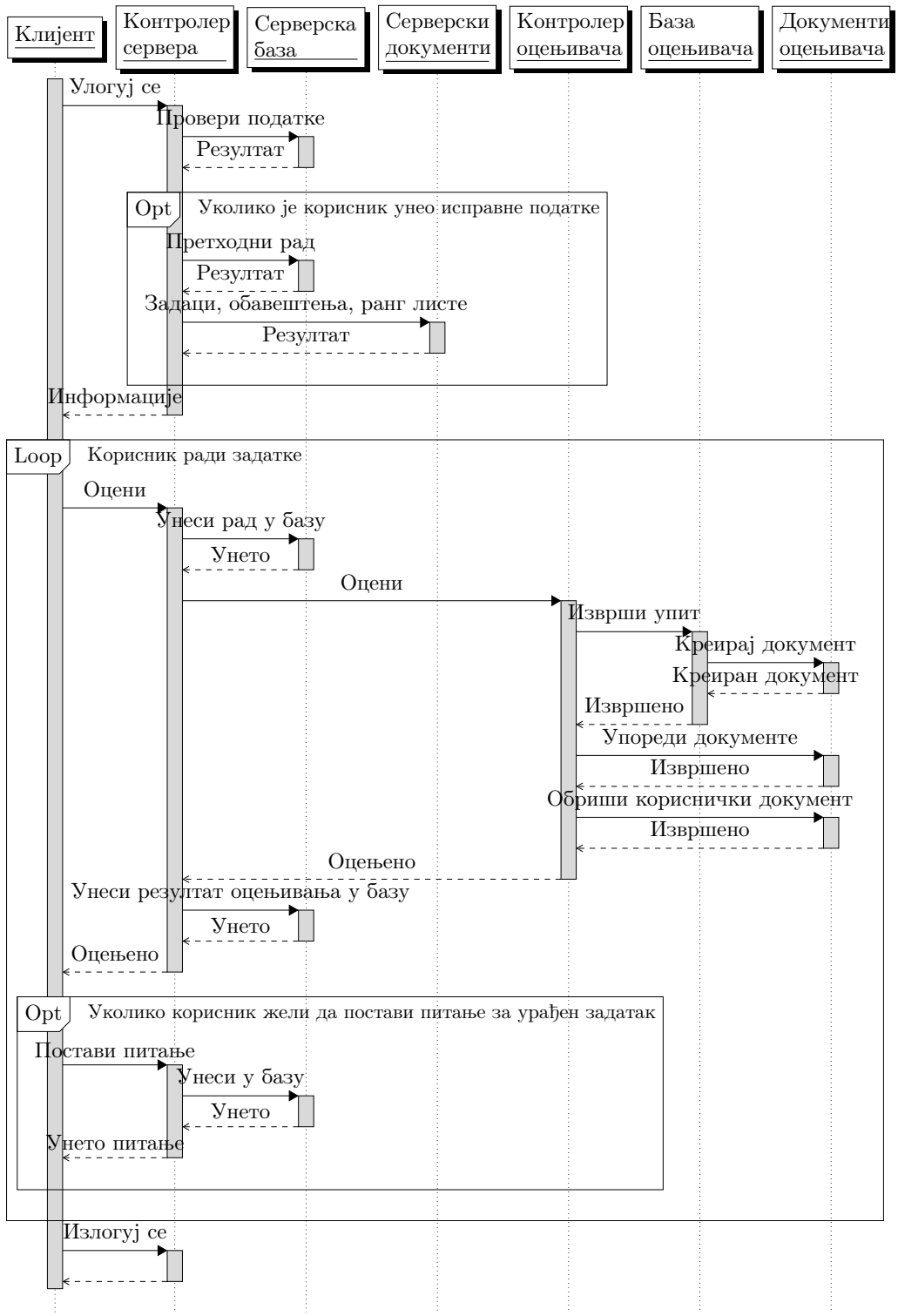
- Оцењивање задатка,
- Оцењивање групе задатака,
- Додавање новог или измену постојећег задатка.

4.1.2 Ток рада апликације

На дијаграму 4.3 приказан уобичајен ток рада система када корисник ради задатке, типично за студентски налог.

Коришћење система од стране студента се састоји из наредних корака:

- Уколико корисник поседује налог, клијент шаље захтев за логовање. Уколико налог не постоји, корисник шаље захтев за прављење налога.
- Уколико је корисник унео исправно корисничко име и шифру, логовање је успешно. Сервер одговара на успешно логовање са информацијама о кориснику (име, презиме, мејл адреса), задацима које он може да ради, задацима које је претходно радио, обавештењима и тренутном ранг листом.
- Процес оцењивања задатака се састоји из наредних корака:
  - Када корисник уради задатак он га шаље серверу кликом на дугме.



Слика 4.3: Дијаграм уобичајеног рада система када га користи студент

  - Сервер прима захтев за оцењивање и он прво у базу уноси захтев корисника за оцењивање. Сервер затим проналази који оцењивач је задужен за оцењивање тог задатка, формира захтев за оцењивање корисничког рада и прослеђује га одговарајућем оцењивачу.
  - Оцењивач прима захтев за оцењивање рада. Он извршава кориснички упит и пореди његов резултат са очекиваним резултатом. На основу резултата поређења он формира одговор и прослеђује га серверу.
  - Сервер прима одговор оцењивача и уноси у базу да ли је корисник успешно решио задатак, а затим прослеђује клијенту одговарајућу поруку.

- Након што је покушао да реши задатак, без обзира на успешност решења, корисник има могућност да постави питање наставнику на следећи начин:
  - Корисник уноси питање, од понуђених наставника бира коме ће питање бити прослеђено, и шаље га серверу кликом на дугме.
  - Сервер уноси постављено питање у базу, након чега ће наставник моћи да га види и одговори.
- На крају рада корисник се излогује.

На дијаграму 4.4 приказан уобичајен ток рада система када га користе наставник или администратор. Коришћење система од стране наставника или администратора се састоји из наредних корака:

- Уколико корисник поседује налог, клијент шаље захтев за логовање. Уколико налог не постоји корисник прави налог.
- Уколико је корисник унео исправно корисничко име и шифру, логовање је успешно. Сервер одговара на успешно логовање са информацијама о кориснику (име, презиме, мејл адреса), задацима које он може да види, обавештењима и тренутном ранг листом.
- Процес уноса нових задатака или измене постојећих се састоји из наредних корака:

  - Корисник уноси измене заједно са упитом који представља једно тачно решење унетог задатка и све то шаље серверу кликом на дугме.
  - Сервер прима захтев за измену или унос задатка и бележи задатак у базу. Сервер затим проналази који оцењивач је задужен за тај задатак, формира захтев за генерисање резултата упита који је примио и прослеђује га одговарајућем оцењивачу.
  - Оцењивач прима захтев за генерисање резултата. Он извршава добијени упит и резултат извршавања упита прослеђује у одговарајући документ. Тај документ ће се користити за оцењивање корисничких радова у будуће.
  - Након што сервер добије одговор од оцењивача да је решење задатка успешно извршено и резултат генерисан и уписан у документ, потребно је ажурирати документ са доступним задацима. Ажурирање се врши тако што се доступни задаци дохвате из базе и унесу у документ у одговарајућем формату.

- Корисник има могућност да одговара на добијена питања на следећи начин:
  - Када корисник унесе одговор на питање, кликом на дугме шаље серверу захтев за унос одговора.
  - Сервер прима овај захтев и уноси одговор у базу. Корисник који је поставио питање ће након овога моћи да види одговор.

- Корисник има могућност да поставља обавештења на следећи начин:
  - Корисник кликом на дугме шаље захтев за постављање новог обавештења са одговарајућим текстом.
  - Сервер прима захтев и уноси ново обавештење у базу. Затим ажурира документ са обавештењима тако што дохвати одређени број обавештења из базе и уноси их у документ. Тај документ садржи одређени број најновијих обавештења која се прослеђују корисницима при логовању.

- Ови корисници могу да раде задатке исто као и студенти, на начин као што је претходно описано.

- На крају рада корисник се излогује.

Ова архитектура доприноси безбедности система. Наиме, корисници прво шаљу захтеве серверу. На серверу се налази база са задацима и корисницима која ни под којим околностима не сме бити угрожена захтевима корисника. Уколико би се кориснички упит извршавао на серверу постојала би могућност да корисник напише упит којим би дохватио тачно решење које се налази у бази задатака, или да промени задатке, обрише друге кориснике. Сервер ће кориснички упит проследити одговарајућем оцењивачу, где ће се исти и извршити. Сваки сервис ради у засебном контејнеру. Према томе, када се извршава упит за оцењивање неће постојати могућност да се изврши повезивање на погрешну базу. База над којом се раде задаци је додатно заштићена као што је описано у делу 5.3.

Сви сервиси могу да се извршавају на физички раздвојеним машинама.

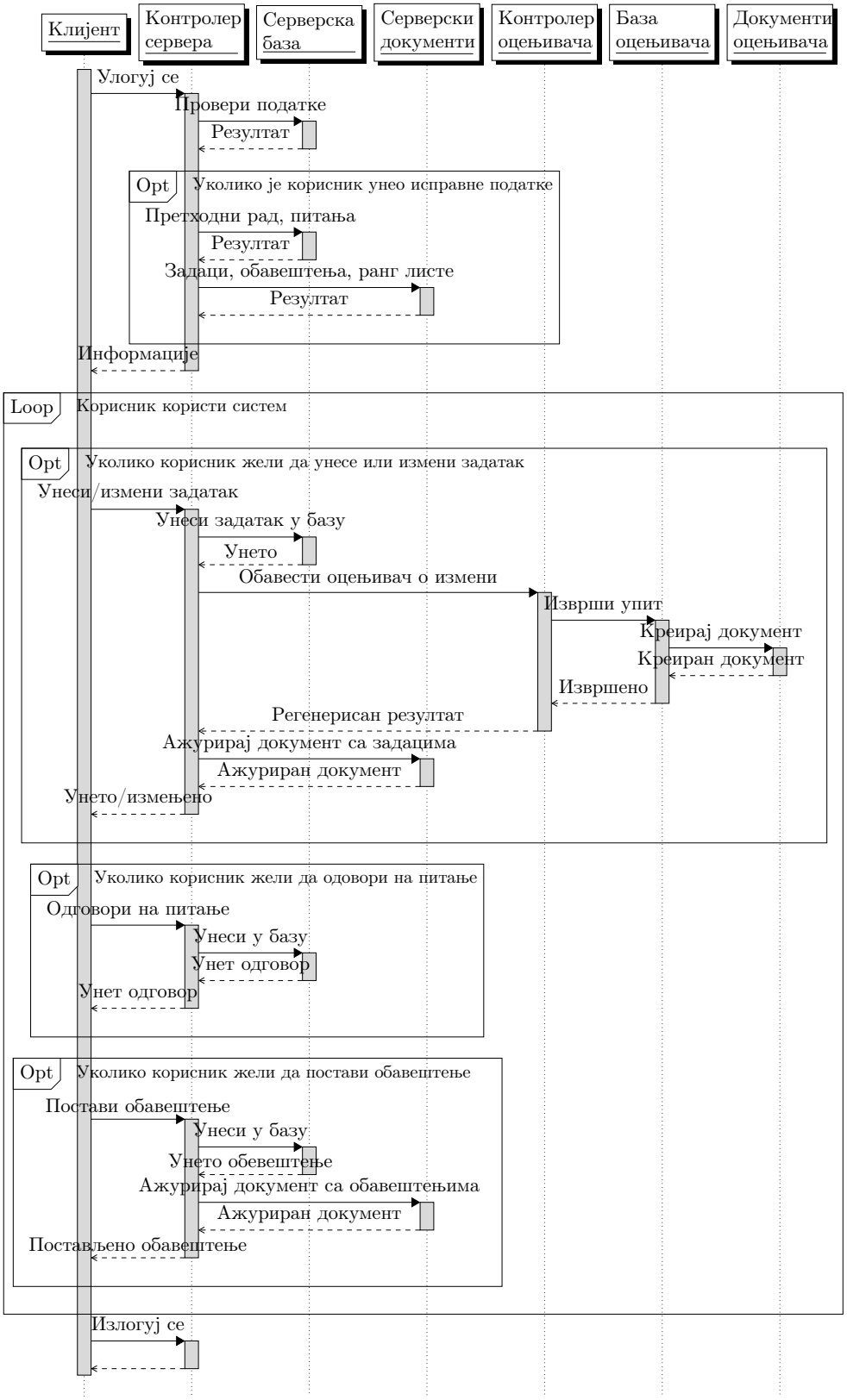
4.2 Технологије

Детаљна имплементација је описана у наредним поглављима, а овде ће бити дата само иницијална идеја зашто се користе одређене технологије.

Систем за управљање базом података је потребан и оцењивачу и серверу. У оба случаја је коришћен систем *Db2* [2]. Поред свих добрих карактеристика овог СУБП које су претходно наведене, додатни разлог да се баш он користи у имплементацији је тај што се користи приликом учења о релационим базама података на Математичком факултету.

Сервер и оцењивач су имплементирани у језику *Java* јер он нуди оно што је у овом раду било потребно, а то је да постоји добра подршка за рад са базама података, нитима, као и могућност једноставног извршавања *Shell* команди.

При оцењивању је потребно извршити неколико команди, између осталог повезивање на базу података, извршавање упита, поређење докумената и више провера да ли је дошло до неке од могућих грешака. У ове сврхе оцењивач користи *shell* скрипте. Уколико би се свака од ових команди извршавала из *Java* апликације оцењивача код би се додатно закомпликовао непотребним понављањима иницијализација сваке операције. Знатно је једноставније све команде обухватити скриптом, а затим ту скрипту извршавати из *Java* апликације за оцењивање. Додатна предност овог приступа је ефикасност. Наиме,



Слика 4.4: Дијаграм уобичајеног рада система када га користи наставник или администратор

можемо замислити да је потребно упоредити два велика документа са резултатима извршавања упита. Један од најефикаснијих начин да се то уради је коришћењем системске *diff* команде, а како се користи скрипта није потребно додатно учитавати велике документе у програму.

Још један разлог за коришћење скрипти је тај што скрипта може да се модификује без заустављања оцењивача и додатне рекомпилације. У скрипти ће се наћи ограничења за извршавање једног упита, то подразумева ограничење времена и системских ресурса. Да би се та ограничења променила биће довољно да се измене само вредности у скрипти која је описана у поглављу 5.4.3. Оцењивач ће наставити да користи ту нову скрипту са измењеним ограничењима. Промена ограничења може бити погодна уколико желимо да повећамо временско ограничење, на пример уколико је сервер добио додатна задужења која га успоравају.

Сервер и оцењивач имају свако своју базу података - оцењивач има базу над којом се извршавају задаци, а сервер има базу са корисницима, задацима, статистикама и осталим табелама које су потребне за функционисање система. У програму оцењивача се бази приступа тако што се користе скрипте које ће извршавати директно повезивање на базу и извршавање упита, као што је претходно речено. У овом случају сама *Java* апликација нема информацију о структури базе и не приступа подацима директно. Са друге стране, из серверске апликације је потребно директно приступати подацима из базе, па она мора имати информацију о структури базе. У овом случају се користи алат *Hibernate* који омогућава угоднији рад са базом података. На пример, при логовању *Hibernate* дохвата све релевантне информације о кориснику, попут задатака које је претходно радио и питања која је поставио, у чему се огледа предност објектно релационог мапирања.

За имплементацију клијентске апликације је коришћен радни оквир *Angular*. Овај оквир је изабран јер омогућава креирање брзих и скалабилних веб апликација.

Као што је претходно наведено, архитектура система подразумева изолованост компоненти. То је постигнуто коришћењем алата *Docker* [4].

Физички сервер представља хардверску компоненту која има свој оперативни систем и хостује апликације. Он је добар када је потребно хостовати једну апликацију и да она буде изолована од других система, што доприноси сигурности.

Виртуелни сервер је софтверски базиран и симулира физички сервер. За разлику од физичког сервера, виртуелни омогућава покретање више независних машина. Овај систем је добар због своје скалабилности и ефикасности. Због тога је изабран као сервер за систем чију имплементацију описује овај рад.

Глава 5

Имплементација оцењивача

Оцењивач је сервис који има за циљ да кориснику одговори да ли је његов код тачно решење датог задатка. У наставку ће бити прво наведена анализа проблема до којих може доћи при оцењивању, а затим и сама имплементација.

Као што је представљено у делу 4.2, оцењивач је имплементиран у језику *Java*, уз коришћење *shell* скрипти. Систем за управљање базом података који се овде користи је *Db2*. Уз мање измене систем може да се прилагоди другом СУБП.

У наставку ће бити прво представљена проблематика задатака који се оцењују, а затим и база која ће бити коришћена за извршавање задатака. Након тога ће бити представљене скрипте за рад са базом и за спровођење процеса оцењивања. Биће описана имплементација контролера оцењивача - *Java* апликације која прима захтеве и позива претходне скрипте. На крају ће бити описано како је извршено тестирање оцењивача.

5.1 Анализа задатака за оцењивање

Тестирање оцењивача је рађено током имплементације, али има смисла на почетку систематично навести који су се проблеми јављали при оцењивању и како су они превазиђени, јер је коначан облик оцењивача добијен након анализе ових проблема и имплементације решења којим се ови проблеми превазилазе.

Што су захтеви задатка комплекснији то може постојати више начина да се исти реши. На пример, један задатак се може решити са подупитима или без њих, са ни једном, једном или више помоћних табела.

У наставку рада ћемо сматрати да је *решење* задатка упит, док ће *резултат* представљати излаз који добијемо извршавањем упита.

Како директно аутоматско поређење упита није могуће, долази се до закључка да упит мора да се изврши. Потребно је извршити поређење резултата извршавања тачног и корисничког упита.

Проблем са овим једноставним приступом је што два упита могу да представљају тачно решење, а да се излази, односно резултати, разликују. На пример, уколико се у задатку не тражи сортирање, тачна решења могу да дају различите резултате. Овај проблем би могао да се реши тиме што би се у сваком задатку тражило сортирање, али то делује мало сувишно. Друга идеја је да корисничким упитима додамо сортирање пре извршавања уколико се оно није тражило у задатку. Додатни проблем везан за ову тематику је што сортирање тражено у задатку може бити недовољно, односно да коначни редослед редова у резултату са задатим сортирањем није јединствен. У овом случају је потребно сортирати бар по још једној колони да би редослед био јединствен.

На пример, у упиту се траже име, презиме, индекс студента и назив предмета који је студент положио, и нека је задато сортирање по имену. У табелама 5.1 су дата два тачна решења са различитим резултатима.

Табела 5.1: Сортирање само по имену

име	**презиме**	**индекс**	**предмет**
Алиса	**Јовановић**	1	п1
Алиса	**Петровић**	2	п2
Бобан	Марковић	3	веб
Бобан	Марковић	3	базе
Цаца	Јанковић	4	веб

име	**презиме**	**индекс**	**предмет**
Алиса	**Петровић**	2	п2
Алиса	**Јовановић**	1	п1
Бобан	Марковић	3	базе
Бобан	Марковић	3	веб
Цаца	Јанковић	4	веб

Сортирање по имену није довољно да јединствено уреди редове резултата. Ни додавање презимена не би било довољно јер може постојати више студената истог имена и презимена. Сортирање по индексу би било довољно да се јединствено поређају студенти, и то је приказано у табелама 5.2.

Табела 5.2: Сортирање само по индексу

име	**презиме**	**индекс**	**предмет**
Алиса	Јовановић	1	п1
Алиса	Петровић	2	п2
Бобан	Марковић	3	веб
Бобан	Марковић	3	**базе**
Цаца	Јанковић	4	**веб**

име	**презиме**	**индекс**	**предмет**
Алиса	Јовановић	1	п1
Алиса	Петровић	2	п2
Бобан	Марковић	3	веб
Бобан	Марковић	3	**веб**
Цаца	Јанковић	4	**базе**

Проблем јединственог сортирања у овом случају остаје јер један студент може да положи више предмета, који се могу наћи у различитим редоследима. Према томе, потребно је додати и сортирање по називу предмета. Дакле, у претходном примеру је неопходно додати сортирање по индексу и називу предмета да би се осигурало да сви тачни упити дају исти излаз. Јединствени излаз који се тражи у овом задатку је дату у табели 5.3.

Табела 5.3: Сортирање по индексу и предмету

име	**презиме**	**индекс**	**предмет**
Алиса	Јовановић	1	п1
Алиса	Петровић	2	п2
Бобан	Марковић	3	веб
Бобан	Марковић	3	базе
Цаца	Јанковић	4	веб

Довољно сортирање зависи од самог задатка. Један од начина да се дода довољно сортирање без претераног размишљања је да се сортира по свакој колони, али то је неефикасно. Овај проблем ће бити решен чувањем додатне информације о томе које сортирање треба додати.

Разликоваћемо три случаја додавања сортирања:

- Уколико задатак захтева јединствено сортирање приликом извршавања, није потребно додавати сортирање. На пример, уколико се траже одређене информације о студенту (име, презиме, смер), и захтева се сортирање по индексу. Корисник мора да изврши сортирање, у супротном задатак није тачан.
- Уколико се у задатку не тражи сортирање, оцењивач треба да дода целокупно сортирање.

- Трећи случај подразумева да се у задатку тражи сортирање, али да то сортирање није довољно за јединствен радослед у резултату. Као и у претходном случају, сортирање се добија из табеле са задацима. Разлика је што се сада очекује да текст за сортирање почиње запетом, на пример „„2,3".

Постоји још један проблем који делом подсећа на сортирање. Уколико имамо задатак у ком се траже парови студената важно је који се студент наводи први, а који други. Не постоји проблем уколико се, на пример, тражи да име првог студента почиње са А а другог са Б. Али уколико се траже различити парови студената који су положили исти број испита, онда је потребно у задатку навести услов којим се одређује који ће се студент наћи на првом, а који на другом месту.

Још један разлог да се резултати извршавања тачних упита разликују јесу називи колона. Уколико се у упиту не наведу називи колона постоји неколико начина да колоне добију називе:

- У једноставном упиту попут наредног називи колона ће бити наслеђени из табеле. Односно колоне ће имати називе IME, PREZIME.

```
1 SELECT IME, PREZIME
2 FROM STUDENT
```

- У наредном примеру ће прва колона, IME, наследити назив из табеле, док ће друга колона за свој назив добити редни број *2*.

```
1 SELECT IME, COUNT(*)
2 FROM STUDENT
3 GROUP BY IME
```

- У претходном једноставном задатку може доћи до проблема уколико корисник одлучи да искористи помоћну табелу. У помоћној табели све колоне морају имати назив, према томе колона са *COUNT(*)* може бити

названа BR. Тај назив ће бити наслеђен у главном делу упита, те ће називи колона на крају бити IME, BR.

```
1  WITH TMP AS(
2          SELECT IME, COUNT(*) BR
3          FROM STUDENT
4          GROUP BY IME)
5  SELECT IME, BR
6  FROM TMP
```

Претходни примери су доста једноставни, али може се замислити да ће корисници у компликованијим задацима користити помоћне табеле, те да ће колоне добијати најразличитија имена. Један начин да се овај проблем реши би био да се игноришу називи колона у свим примерима, што би нарушавало смисао задатака у којима се траже називи колона. Уколико се не траже називи свих колона јако би било тешко додати назив неким колонама. Наиме, уколико замислимо линију текста са речима које представљају називе колона, није лако игнорисати само неке од тих речи у поређењу.

На пример, уколико се у задатку траже име, број положених испита и назив смера студента, а при том прва колона треба имати назив *име*, последња *смер*, средња остаје без назива. Очекиваће се наредни излаз:

```
IME                2                SMER
- - - - - - - - - - - - - - - - - - - - - - - - - -
```

Уколико је корисник искористио помоћну табелу да преброји испите, и тој колони дао назив *BR*, добија се наредни излаз:

```
IME                BR                SMER
- - - - - - - - - - - - - - - - - - - - - - - - - -
```

У овом случају није једноставно занемарити назив само једне колоне јер би било потребно проверити прву и трећу реч. Додатно треба узети у обзир

да назив колоне може да се састоји из више речи, па назив смера можда не буде на трећем месту.

Могуће је имплементирати оцењивач у ком би се за неке задатке проверавали називи колона, а за неке не. У том случају би се у табели са свим задацима морали чувати индикатори да ли треба проверавати називе колона. У оцењивачу чија ће имплементација бити представљена у наставку ће се захтевати да у сваком задатку мора да се тражи експлицитно од корисника да именује све колоне и да их наведе у редоследу у ком су наведене у задатку.

Током тестирања је уочен проблем са задацима у којима се тражи тренутни датум. Као што је поменуто у делу 4, оцењивање се врши тако што се резултат извршавања корисничког упита пореди са резултатом извршавања тачног упита, а који се налази у документу и који је већ креиран некада раније. Проблем је што упит из већ постојећег документа можда није извршен истог дана када се задатак оцењује. Према томе, добијени су различити резултати јер се тренутни датуми разликују. Идентичан проблем постоји и у задацима у којима се тражи тренутно време.

Једна идеја за решење проблема за датум је да се тачан упит изврши једном дневно, односно када се промени датум. Тиме би нови документ са резултатом могао да се користи за оцењивање целог дана. Или још боље, да се тачан упит изврши први пут у току дана када се тај задатак оцењује. Овим би се избегло извршавање упита оних дана када нико не решава тај задатак.

Друго решење би било да се константа за тренутни датум, односно време замени неком конкретном вредношћу у упиту. На пример, да се *current date* замени са *01.01.2024.* Како ће се и тачан и кориснички упит извршити са овом фиксном вредношћу неће бити потребно извршавати тачан упит при промени датума.

Проблем тренутног времена је нешто тежи јер би кориснички и тачан упит морали да се изврше у исто време, или у веома малом временском размаку. За решавање овог проблема је коришћен приступ сличан претходном, односно константе *current time* и *current timestamp* се мењају неким унапред одређеним вредностима.

Наредна ситуација није нужно проблем, већ нешто што треба имати у виду при додавању задатака за оцењивач. Тачно решење треба да се изврши у разумном времену. На пример, нека имамо табелу са 119 144 редова (као на пример табела испита у бази *студ2020* која је описана у 5.2) и табелу

са 3496 редова (као на пример табела студената у бази *студ2020*). И нека имамо задатак који треба да представља пример слободног спајања, односно да је потребно спојити сваки ред једне табеле са сваким редом друге табеле. Када искористимо табелу испита и студената добићемо 416 527 424 редова, а упит ће се извршавати преко једног минута. Много би било боље идентичну идеју приказати мањим табелама, и сличним задатком чије ће се решење извршавати знатно брже.

Кориснички упити могу дуго да се извршавају, па је потребно поставити ограничење за њихово извршавање. Овим ће се избећи да неефикасан кориснички упит превише дуго заузима ресурсе. Идеално би било да граница извршавања тачног решења не буде близу задатог ограничења.

Да сумирамо, задатак који се оцењује треба да испуни наредне услове:

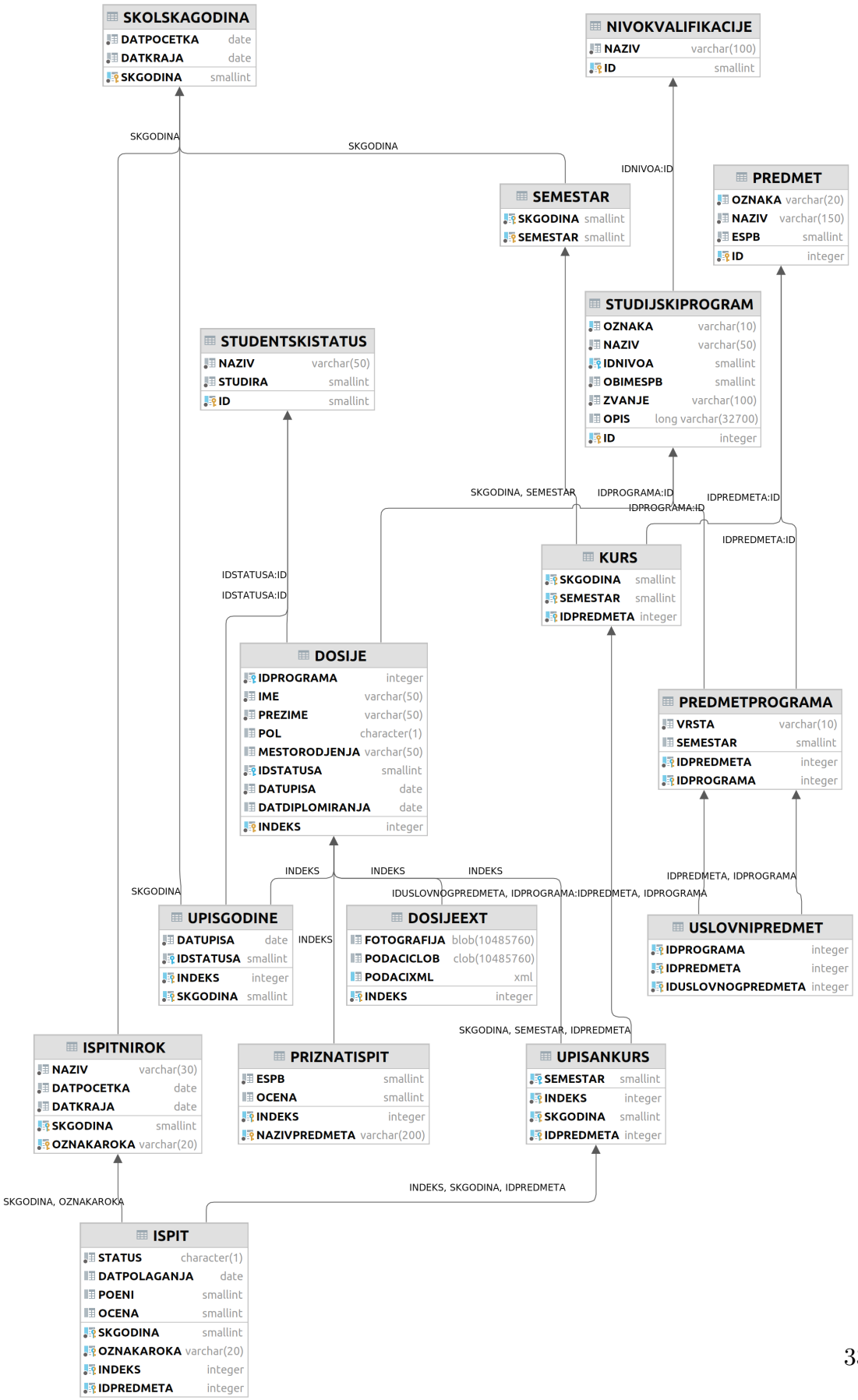
- Задатак мора имати сортирање које јединствено одређује редослед редова у резултату. То сортирање може бити део задатка или додато при извршавању.
- У задатку мора да се тражи навођење назива свих колона.
- Тачно решење треба да се извршава у разумном времену.

5.2 База коришћена за оцењивач

База коришћена за оцењивач је база над којом се раде задаци, односно у задацима ће се тражити да се издвоје подаци из ове базе. Базу *студ2020* користе студенти Математичког факултета за учење *SQL*-а на трећој и четвртој години студија. Због тога је управо ова база изабрана за коришћење у оцењивачу.

На слици 5.1 је представљена структура базе. База има за циљ да представи информације о студентима на факултету. База поседује две схеме, *DA* и *DB*. Ове схеме поседују идентичне табеле, разлика је што су подаци у првој записани латиницом, док су у другој записани ћирилицом. У овом раду се користи само схема *DA*. Табела *DOSIJE* садржи основне информације о студенту, индекс као примарни кључ, име, презиме, место рођења студента и друго. Сваки студент студира неки од смерова који се налазе у табели *STUDIJSKIPROGRAM*. Сваке године студент уписује школску годину

и то је представљено табелом *UPISGODINE*. Када упише годину студент уписује предмете које жели да похађа те године, што је представљено табелом *UPISANKURS*. Табела *KURS* даје информацију да се у датој години одржавају часови из предмет из табеле *PREDMET*. Студент ће током године коју је уписао полагати предмете које је уписао и та полагања ће бити забележена у табели *ISPIT*.



Слика 5.1: Схема базе *студ2020*

5.3 Додатни захтеви за базу

Корисници треба да имају само дозволу за извршавање упита који дохватају редове из базе. Корисник не сме да има могућност да на било који начин мења базу, да креира и брише табеле, односно да додаје и брише редове. Разлог за ово је тај што се сви кориснички упити извршавају над истом базом. Уколико сваки корисник буде произвољно мењао табеле добијаће се различити резултати у зависности од стања табела.

Изузетак из претходно наведеног је корисник који је администратор. Он има право да измени базу, али да при томе води рачуна да систем остане конзистентан. У наставку је наведено како се база штити од корисника, с тим да се сматра да администратор има виша права у систему па се на њега то не односи.

При креирању базе у докер контејнеру креира се корисник *db2inst1*. Како тај корисник уједно представља администратора, он има сва права над базом. Са друге стране, кориснички упити не смеју да се извршавају са привилегијама администратора.

Како се прављење новог *db2* корисника у докер контејнеру показало као захтевно одабран је други приступ. У докер контејнеру је направљен нови системски корисник, *student*. Тај корисник има приступ групе *public*, па је у бази додата рестрикција за управо ту групу.

Додавањем наредне линије у докер фајл креира се нови корисник са називом *student* и шифром *matf2024.*:

```
RUN useradd -m -s /bin/bash student && \
        echo 'student:matf2024.' | chpasswd
```

Када се системски корисник повеже на базу он се подразумевано налазу у истоименој схеми. Због тога је потребно направити схему *student* и ограничити права над њом. Наиме, у новонаправљеној схеми није дозвољено креирати, брисати и мењати табеле. То се постиже наредним командама:

```
create schema student;
revoke alterin on schema student from public by all;
```

```
3  revoke createin on schema student from public by all;
4  revoke dropin on schema student from public by all;
```

Поред укидања права на измене нове схеме, потребно је иста права уклонити и на већ постојећој схеми *DA*:

```
1  revoke alterin on schema da from public by all;
2  revoke createin on schema da from public by all;
3  revoke dropin on schema da from public by all;
```

Једино право које *student* треба да има је да приступа редовима табеле, то се постиже тако што се изврши наредна команда за сваку табелу базе:

```
1  grant select
2  on naziv_tabele
3  to public;
```

Уколико корисник покуша да обрише табелу, креира или измени табелу на било који начин добиће одговарајућу поруку о недостатку права. На пример, уколико покуша да обрише табелу *gocuje* у схему *DA* добиће наредну поруку:

```
DB21034E  The command was processed as an SQL statement because it was
↪  not a
valid Command Line Processor command.  During SQL processing it
↪  returned:
SQL0551N  The statement failed because the authorization ID does not
↪  have the
required authorization or privilege to perform the operation.
↪  Authorization
ID: "STUDENT".  Operation: "DROP TABLE". Object: "DA.DOSIJE".
↪  SQLSTATE=42501
```

Администратор једини има права да мења базу над којом се раде задаци. Уколико промени базу, он мора водити рачуна да поново изврши упите који представљају тачна решења да би систем наставио да ради исправно.

5.4 Скрипте оцењивача

Оцењивач користи три скрипте, прву за генерисање резултата на основу датог тачног решења задатка, другу за оцењивање прослеђеног рада и трећу за прекидање свих упита који прелазе постављена ограничења. Прве две скрипте се позивају из контролера, док се трећа скрипта покреће на почетку, када и контролер. Као што је поменуто у поглављу 5.1, упит који представља тачно решење се извршава једном при уносу задатка у базу. За генерисање резултата тачног решења користи се скрипта која је описана у наставку.

5.4.1 Генерисање резултата на основу датог тачног решења

У наставку ће бити наведене и образложене команде скрипте.

Документ који скрипта очекује на улазу садржи идентификаторе задатака и упите које треба извршити. Сваки задатак у улазном документу треба да буде представљен са два реда, идентификатор у једном и упит у другом. Скрипта ће за сваки задатак у улазном документу креирати по један документ са називом који одговара идентификатору задатка, и у тај документ уписати резултат извршавања упита који се односи на тај задатак.

Прво је потребно наредном командом проверити да ли је при покретању скрипте прослеђен аргумент:

```
1 if [ "$#" -ne 1 ] ; then
2         echo "Usage: file"
3         exit 1
4 fi
```

Наредним се проверава да ли постоји документ са прослеђеним називом:

```
5 if [ ! -e "$1"  ]; then
6         echo "Server error | File does not exist"
7         exit 1
8 fi
```

Очекује се да је прослеђени претходно описан документ. Уколико документ недостаје потребно је исписати одговарајућу поруку и прекинути извршавање.

Документ са наредним садржајем представља валидан улаз за скрипту:

```
1
SELECT * FROM DA.DOSIJE
2
SELECT * FROM DOSIJE
```

Прве две линије представљају први задатак, 1 у првој линији представља идентификатор, док се у другој линији налази упит. Друге две линије представљају други задатак. Упит другог задатка није тачан јер не постоји табела DOSIJE ван схеме DA. Резултат извршавања скрипте са овим фајлом на улазу ће бити представљен нешто касније.

Наредном командом се врши повезивање на базу података *студ2020*:

```
9  connection_result="$(db2 connect to stud2020)"
10
11 if [[ $connection_result != *Database\ Connection\ Information*
   ↪  ]]; then
12         echo "Server error | Unable to connect to database"
13         exit 1
14 fi
```

Излаз команде за повезивање на базу је додељен промењливој *connection_result*. Први разлог за то је да би се проверило да ли је повезивање било успешно, што се и ради у једанаестој линији. Други разлог је што се као излаз скрипте очекује само резултат извршавања упита, што ће бити описано у наставку. Додатни излаз би нарушио тај формат.

Петља чита две по две линије документа који је прослеђен као аргумент:

```
15 while read -r task_id; do
16         read -r query
```

```
17          start=$(date +%s.%N)
```

Вредност прве од две прочитане линије се додељује промењивој *task_id* и представља идентификатор задатка. Друга линија се додељује промењивој *query* и представља упит који треба извршити. Иницијализује се време почетка извршавања.

У наставку је извршавање упита:

```
18          set -o pipefail
19          db2 "$query" | sed '/^$/d' > "results/$task_id"
20          if [ $? -ne 0 ]; then
21                  echo "$task_id#Time, CPU or cost estimate limit
                    ↪  exceeded"
22                  connection_result="$(db2 connect to stud2020)"
23                  if [[ $connection_result != *Database\
                    ↪  Connection\ Information*  ]]; then
24                          echo "Server error | Unable to connect
                            ↪  to database"
25                          exit 1
26                  fi
27                  continue
28          fi
```

Команда *db2* на линији 19 се користи за извршавање упита који се налази у промењивој *query*. Њен излаз се прослеђује команди *sed* која уклања празне линије. Излаз команде *sed* се прослеђује у одговарајући документ, односно документ са називом из промењиве *task_id* у директоријуму *results*. Након што се извршавање команде заврши проверава се да ли је извршавање било успешно. Како се извршавање на линији 19 састоји из више команди, од којих било која може да произведе грешку, командом на линији 18 је укључена опција да је извршавање уланчаних команди неуспешно ако је бар једна команда неуспешна. Извршавање упита може бити насилно прекинуто од стране процеса који води рачуна о ресурсима које сваки упит користи при извршавању, а о ком ће бити речи нешто касније. У овом случају се исписује

порука да упит користи превише ресурса. Уколико је извршавање упита било насилно прекинуто, и конекција на базу ће бити прекинута па ју је потребно поново успоставити (линија 22) да би наредни упити из документа могли да се изврше. Повезивање на базу података се врши као на почетку ове скрипте.

Уколико је претходно успешно извршено проверава се да ли је документ креиран:

```
29          if [ ! -e "results/$task_id"  ]; then
30                  echo "$task_id#File is not created "
31                  continue
32          fi
```

Проверава се да ли је упит успешно извршен:

```
33          last_line=$(tail -1 "results/$task_id")
34          if ! echo "$last_line" | grep -qE '[0-9]+ record\(s\)
            ↪  selected\.'; then
35                  echo "$task_id#Number of rows missing"
36                  continue
37          fi
```

У систему *db2* се на крају излаза успешног извршавања упита очекује ред наредног формата:

```
1234 record(s) selected.
```

Уколико број редова недостаје на стандардни излаз се исписује порука и прелази се на наредни задатак.

```
38          end=$(date +%s.%N)
39          duration=$(echo "$end - $start" | bc)
40          number_of_rows=$(echo "$last_line" | sed
            ↪  's/[^0-9]*\([0-9]\+\).*/\1/')
```

```
41          echo "$task_id#$duration#$number_of_rows"
42  done < $1
```

Уколико је упит успешно извршен на стандардни излаз ће бити исписан текст у наредном формату:

`task_id#$duration#$number_of_rows`

У претходном испису се користе наредне промењиве:

- *task_ id* - идентификатор задатка,
- *duration* - време извршавања датог упита,
- *number_ of_ rows* број редова који су добијени извршавањем.

Након што петља прочита све редове из документа потребно је извршити још неколико команди:

```
43  end_x=$(date +%s.%N)
44  duration=$(echo "$end_x - $start_x" | bc)
45  echo "$duration"
46
47  connection_result="$(db2 connect reset)"
48  if [[ $connection_result != *completed\ successfully*  ]]; then
49          echo "Server error | Error while disconnecting"
50          exit 1
51  fi
52
53  rm "$1"
54
55  exit 0
```

Промењивој *end_ x* се додељује тренутно време. Затим се рачуна колико је трајало укупно извршавање скрипте и то се исписује на стандардни излаз. Прекида се конекција са базом командом на линији 47. Излаз ове операције

се преусмерава у промењиву *connection_result* и проверава, слично као код конектовања. На крају је потребно обрисати документ који је прослеђен као аргумент.

Покретањем скрипте са документом наведеним на почетку се добија наредни излаз:

```
1#.329720592#3496
2#Number of rows missing
.381605370
```

Приликом генерисања решења није вршена додатна паралелизација јер је сматрано да се генерисање великог броја решења неће вршити често. Уколико би се паралелизовало било би потребно за свако извршавање успоставити додатну конекцију на базу.

5.4.2 Оцењивање задатака

У наставку ће бити наведени кораци скрипте за проверу решења.

Скрипта ће прво проверити аргументе:

```
1 if [ "$#" -ne 3 ] ; then
2         echo "Usage: user_path solution_path query"
3         exit 1
4 fi
```

Скрипта очекује три аргумента: путању до документа у који треба да се упише резултат извршавања корисничког упита, путању до документа са тачним резултатом и кориснички упит који треба оценити.

Уколико је прослеђен одговарајући број аргумената, они се додељују одговарајућим промењивама:

```
5 user_path="user_results/$1"
6 solution_path="results/$2"
7 query="$3"
```

Скрипта проверава да ли документ са тачним резултатом постоји и прекида извршавање уколико не постоји:

```
8  if [ ! -e "$solution_path"  ]; then
9          echo "Server error | Result of solution for task is
           ↪  missing"
10         exit 1
11 fi
```

Повезивање на базу и мерење времена извршавања се врши као и у претходној скрипти:

```
12 connection_result="$(db2 connect to stud2020 user student using
   ↪  matf2024.)"
13 if [[ $connection_result != *Database\ Connection\ Information*
   ↪  ]]; then
14         echo "Server error | Unable to connect to database"
15         exit 1
16 fi
17 start=$(date +%s.%N)
```

За разлику од претходне скрипте, овде се за повезивање користи корисник *студент* са ограниченим правима приступа. Разлог томе је што наставници имају виши ниво привилегија, па ће се за извршавање њихових упита користити корисник *db2inst1*. Креирање корисника *студент* је објашњено у поглављу 5.3. Извршавање упита се врши командом *db2*, чији се излаз прослеђује команди *sed* која уклања празне редове и сувишне белине, а затим се тако обрађен узлаз прослеђује у одговарајући документ.

Извршавање упита се врши на следећи начин:

```
18 set -o pipefail
19 db2 "$query" | sed '/^$/d' > "$user_path"
20 if [ $? -ne 0 ]; then
```

```
21          echo "User error | Time, CPU or cost estimate limit
            ↪  exceeded"
22          if [ -e "$user_path"  ]; then
23                  rm "$user_path"
24          fi
25          exit 1
26  fi
27  end=$(date +%s.%N)
```

Као и у претходној скрипти, и овде се укључује опција која проверава да ли је нека од уланчаних команди проузроковала грешку. Уколико је дошло до грешке највероватније је у питању прекорачење дозвољених ресурса за упит, па се на стандардни излаз исписује одговарајућа порука, а документ са резултатом се брише уколико је креиран.

Уколико претходно није дошло до грешке треба проверити да ли је документ заиста креиран:

```
31  if [ ! -e "$user_path"  ]; then
32          echo "Server error | User file creation failed"
33          exit 1
34  fi
```

У овом тренутку извршавања скрипте постоји документ тачно одређеног назива који садржи резултат извршавања упита који се оцењује. У наставку је потребно извршити поређење овог документа са документом са тачним резултатом. Да би корисник добио што кориснију поруку поређење ће се вршити из неколико делова. За то ће нам бити потребна прва и последња линија новокреираног документа, као и прва и последња линија документа са тачним резултатом. То ћемо добити коришћењем команди *head* и *tail* на следећи начин:

```
35  user_end=$(tail -1 "$user_path")
36  solution_end=$(tail -1 "$solution_path")
```

```
37  user_start=$(head -1 "$user_path")
38  solution_start=$(head -1 "$solution_path")
```

Прве линије садрже називе колона, док последње линије садрже број редова који је добијен извршавањем упита.

Врши се провера прве линије, односно почетка документа:

```
39  if [[ $user_start == DB2* ]] || [[ $user_start == SQL* ]]; then
40          echo "User error | Not executing"
41          rm "$user_path"
42          exit 1
43  fi
```

Уколико кориснички документ, односно прва линија, почиње са *DB2* или *SQL* то имплицира да је дошло до неке грешке при извршавању. Грешка је највероватније синтаксна или корисник нема одговарајућа права, уколико је на пример покушао да креира табелу.

Даље се врши поређење последњих линија:

```
44  if [ "$user_end" != "$solution_end" ]; then
45          echo "User error | Incorrect number of rows"
46          rm "$user_path"
47          exit 1
48  fi
```

Последња линија документа ће садржати број редова добијених извршавањем. Уколико се последње линије докумената који се пореде разликују, онда је број редова добијен извршавањем упита погрешан. У овом случају се добија порука да је погрешан број редова, брише се креирани документ и завршава се извршавање. Овде корисник неће добијати информацију о очекиваном броју редова да би се избегао покушај корисника да вештачки намести резултат.

У овом тренутку извршавања се зна да није дошло до грешке и да је корисник добио одговарајући број редова па има смисла вршити комплекснија поређења. Даље се врши поређење првих линија:

```
49 first_row_diff=$(diff -bBi <(echo "$user_start") <(echo
   ↪  "$solution_start"))
50 if [ "$first_row_diff" ]; then
51         echo "User error | Wrong column names"
52         rm "$user_path"
53         exit 1
54 fi
```

Прва линије документа садржи називе колона. Прве линије докумената се пореде са игнорисањем величине слова и сувишних белина, односно коришћењем команде *diff* са опцијама *-bBi*. Ако се називи колона разликују нема смисла настављати поређење, корисник ће добити одговор да називи колона нису добри, документ ће бити обрисан и извршавање ће бити прекинуто.

Када су прве и последње линије упоређене и све се поклапа има смисла упоредити остатак документа:

```
55 difference=$(diff -bBi <(tail -n +3 "$user_path") <(tail -n +3
   ↪  "$solution_path") | head -n 1)
56 if [ -n "$difference" ]; then
57         echo "User error | Incorrect"
58 else
59         duration=$(echo "$end - $start" | bc)
60         echo "OK | Execution time $duration"
61 fi
```

Командом *diff* се врши поређење. Опције *-bBi* се користе за игнорисање белина као и величине слова. Команда `tail -n +3 "\$user\_path"` узима садржај документ од треће линије до краја. Разлог за ово је што је прва линија већ проверена, док друга линија садржи само цртице па је није потребно поредити. Наиме, излаз извршавања упита је у наредном формату:

```
INDEKS      IME         PREZIME
- - - - - - - - - - - - - - - - - - - - -
123            ...           ...
...
```

Додатни разлог је што друга линија, односно линија са цртицама, може садржати различит број цртица, у зависности од броја белина у остатку документа.

Променљива *difference* ће садржати разлику између докумената. Уколико су документи идентични ова променљива ће бити празна. Да би се избегло чување веће количине линија у променљивој, командом `head -n 1` се узима прва линија разлике. Уколико су документи идентични, та линија ће заправо бити празна, док ако имамо 100 линија разлике у променљивој ћемо имати само прву линија разлике, што нам је довољно да видимо да документи нису исти. У зависности од променљиве *difference* се на стандардни излаз исписује порука да је решење добро, заједно са временом извршавања, или да решење није добро.

На крају је потребно обрисати креирани документ са корисничким резултатом:

```
62 rm "$user_path"
63 exit 0
```

У наставку су сви излази које скрипта може да да:

- *Server error | Unable to connect to database* - повезивање на базу података није успело,
- *Server error | Result of solution for task is missing* - оцењивач нема документ са тачним резултатом задатка. Грешку није изазвао корисник па се исписује порука да је у питању серверска грешка и опис грешке,
- *Server error | User file creation failed* - из неког разлога је дошло до грешке при креирању документа (разлог који би могао изазвати ово ће бити дат у делу 8.1),

- *User error | Time, CPU or cost estimate limit exceeded* - извршавање користи више ресурса него што је дозвољено,
- *User error | Not executing* - велика је вероватноћа да је дошло до синтаксне грешке, обухвата и случај да корисник покуша да изврши команду која му није дозвољена, на пример брисање табеле,
- *User error | Incorrect number of rows* - кориснички упит даје погрешан број редова,
- *User error | Wrong column names* - корисник је дао погрешне називе колонама,
- *User error | Incorrect* - корисник је направио неку грешку која претходно није покривена, односно ако сами редови садрже неку разлику у односу на тачан резултат. На пример ако је сортирање погрешно, ако је формат текста погрешан, погрешан испис и слично,
- *OK | Execution time $duration* - решење је тачно. Корисник ће добити поруку колико је трајало извршавање његовог упита.

Поруке у скрипти могу да се мењају. Једино ограничење је да порука мора да почиње са *User error*, *Server error* или *OK* јер се на основу тога у апликацији препознаје тип грешке. Остатак поруке се може променити да буде на српском или неком другом језику.

5.4.3 Постављање ограничења за извршавање упита

Као што је претходно наговештавано, извршавање сваког упита мора бити ограничено. Један начин да се време извршавања ограничи је командом *timeout*. Команда као аргумент прима временско ограничење и команду коју треба да изврши. Уколико команда која се извршава достигне временско ограничење биће насилно прекинута. Иако се ово добро показало када се гледа време као ресурс, није било довољно када су у питању просторни или процесорски ресурси.

Систем *Db2* у табели *SYSIBMADM.MON_CURRENT_SQL* садржи упите који се тренутно извршавају или чекају на извршавање. Број упита који се извршава је ограничен бројем процесора.

Претходно поменута табела, између осталог, садржи наредне колоне:

- *APPLICATION_HANDLE* - представља јединствени идентификатор тренутне сесије унутар *Db2* система,
- *ELAPSED_TIME_SEC* - представља протекло време извршавања упита у секундама,
- *TOTAL_CPU_TIME* - представља укупно процесорско време утрошено за извршавање упита у милисекундама,
- *QUERY_COST_ESTIMATE* - представља процену цене извршавања упита. Ову вредност процењује систем на основу статистика које садржи. Те статистике се освежавају командом `db2 reorgchk update statistics`, а како се не очекују промене базе података једно извршавање ове команде на почетку рада система је довољно.

Наредна скрипта сваке секунде проверава статус извршавања упита и прекида упите који су заузели превише ресурса:

```
1  db2 connect to stud2020
2  while true; do
3          execution=$(db2 "SELECT APPLICATION_HANDLE FROM
           ↪  SYSIBMADM.MON_CURRENT_SQL WHERE ELAPSED_TIME_SEC>10
           ↪  OR TOTAL_CPU_TIME>30000 OR
           ↪  QUERY_COST_ESTIMATE>400000")
4          ids=$(echo "$execution" | sed "1,3d" | head -n -2 | tr
           ↪  '\n' ',' | head -c -1)
5          if [ "$ids" ]; then
6                  db2 "force application ( $ids )"
7                  echo killed
8          fi
9  sleep 1
10 done
11
```

Претходна скрипта прво успоставља конекцију на базу, а затим покреће бесконачну петљу која ће проверавати стање извршавања упита сваке секунде. На линији 3 се дохватају идентификатори упита који прелазе задата ограничења. Овде се не гледа да ли се упит тренутно извршава, односно да ли је у статусу *executing* или *idle*. Разлог томе је што систем *Db2* поседује механизам који ће привремено паузирати извршавање захтевног упита да би се извршио мање захтеван упит, попут упита на линији 3, у ком случају га ставља у *idle* стање. Променљива *execution* садржи резултат извршавања упита, и он изгледа, на пример, овако:

```
APPLICATION_HANDLE
--------------------
14400
14398

2 record(s) selected.
```

Уланчаним командама на линији 4 ће се садржај ове променљиве трансформисати у наредни облик:

```
14400,14398
```

Команда на линији 6 ће прекинути сесије са прослеђеним идентификаторима, односно прекинути извршавање захтевних упита.

Вредности које су дате као ограничења за упите су одређене тестирањем оптерећења система, што је представљено у делу 5.6.

Овде се може још размотрити да ли ова скрипта треба да се извршава све време или само по потреби. Наиме, уколико се ни један упит не извршава, не мора ни ова скрипта да се извршава. Да би се то обезбедило, контролер оцењивача би требало да има механизам који покреће скрипту пре него покрене неку од две скрипте које извршавају упите, а прекида је након што се извршавање свих упита заврши. Међутим, сама ова скрипта није захтевна јер је број радова који се оцењује ограничен, о чему ће бити речи нешто касније, па је овде имплементирано да скрипта ради све време. Ако оцењивач није заузет оцењивањем, онда извршавање ове скрипте не представља велико оптерећење за цео систем.

Ову скрипту је довољно покренути једном при покретању оцењивача. Покретање више пута, на пример при сваком оцењивању, може направити проблем јер ће више процеса радити исту ствар и потенцијално изазвати проблем са конкурентношћу.

5.5 Имплементација контролера

Као што је претходно помињано, апликација за оцењивање је имплементирана у програмском језику *Java*. Коришћена је библиотека *Grizzly* за руковање HTTP захтевима.

Програмски језик *Java* поседује библиотеку *ProcessBuilder* која омогућава извршавање *shell* команди. Како је потребно извршити више команди, уместо да се креира по један процес за сваку, команде су обједињене скриптом која се извршава у једном процесу. Излаз извршавања скрипте је преусмерен у *Java* апликацију и одатле се прослеђује кориснику.

За сам оцењивач су кључна два захтева: за креирање тачног резултата и за проверу решења.

5.5.1 Генерисање тачног резултата

Захтев за генерисање резултата се шаље POST методом на путању */grader/generateResults* са телом захтева у наредном формату:

```
[
        {
                "taskId": 1,
                "solution": "....",
                "ordering": "1,2,3"
        }
]
```

Елементи претходног захтева су:

- *taskId* - јединствени идентификатор задатка чији се резултат генерише. У апликацији се у ову сврху користи примарни кључ из табеле са задацима која ће бити описана у наставку;

- *solution* - упит који представља тачно решење задатка;
- *ordering* - наставак који треба додати за сортирање, може да се изостави уколико није потребно.

Под претпоставком да ће случајеви када је потребно генерисати само један резултат бити ретки, имплементација подразумева да се може послати више задатака за генерисање резултата у једном захтеву. Када стигне захтев он се обрађује и на основу њега се креира документ са садржајем тачно одређеног формата, као што је објашњено у поглављу 5.4.1. Обрађивање упита се састоји од замене *current date* и *current time* са константним вредностима, као и додавања сортирања.

Документ добија јединствен назив, *generate + јединствен идентификатор*. Јединствен идентификатор представља вредност коју враћа функција `UUID.randomUUID()`. Позива се скрипта за генерисање резултата из дела 5.4.1 којој се као аргумент прослеђује овај документ. Скрипта генерише резултат.

Након генерисања резултата, оцењивач ће обрадити одговор скрипте и послати га као одговор апликацији која је послала захтев. Одговор ће бити у наредном формату:

```
{
"totalTime": ".723",
"tasks": [
        {
                "taskId": "2001",
                "noRows": "989",
                "time": ".151927837"
        },
        {
                "taskId": "2002",
                "error": "Number of rows missing"
        }
   ]
}
```

У претходном примеру одговор ће се састојати од укупног времена извршавања, односно *totalTime*, и низа *tasks* који садржи резултате извршавања сваког од извршених упита. Уколико је упит успешно извршен његов одговор ће бити представљен објектом са наредним пољима:

- *taskId* - идентификатор задатка који је оцењен,
- *noRows* - број редова који је добијен извршавањем упита,
- *time* - време извршавања упита.

Уколико је дошло до грешке одговор за одговарајући задатак ће се састојати од:

- *taskId* - идентификатор задатка,
- *error* - порука о грешци до које је дошло.

5.5.2 Оцењивање задатка

Захтев за оцењивање се шаље на */grader/checkSolution*, и користи се POST метод. Тело захтева мора бити у наредном формату:

```
{
        "requestId": "...",
        "taskId": "1",
        "solution": "...",
        "ordering": "1,2"
}
```

Захтев се састоји од:

- *requestId* - јединствени идентификатор задатка који треба оценити. То је комбинација идентификатора корисника и идентификатора задатка,
- *taskId* - јединствени идентификатор задатка који се оцењује. Користи се примарни кључ из табеле задатака. Као што је наглашено у делу 4, ова вредност је битна јер се у документу са тим називом очекује резултат извршавања тачног решења,

- *solution* - упит који представља корисничко решење,
- *ordering* - наставак који треба додати за сортирање, може да се прескочи као поље уколико није потребно.

Када се овај захтев прими позива се скрипта оцењивача из поглавља 5.4.2, а као улазне аргументе добија податке из тела захтева. На основу резултата извршавања скрипте се формира одговор и прослеђује кориснику. Одговор је у наредном формату:

```
{
        "requestId": "...",
        "status": true,
        "message": "..."
}
```

При чему је:

- *requestId* - јединствени идентификатор задатка који је добијен из захтева,
- *status* - вредност која је *true* уколико је задатак тачно решен, у супротном је *false*,
- *message* - додатна порука о успешности решеног задатка; порука ће бити прослеђена из скрипте: уколико је задатак тачно решен биће време извршавања, иначе ће бити порука која ближе одређује грешку.

Разматране алтернативе за обраду захтева за оцењивање

Оцењивач функционише тако што прима захтеве и обрађује их у одговарајућим функцијама. Повратна вредност тих функција је оно што се враћа кориснику који је послао захтев. Како оцењивање задатка траје извесно време размотрено је неколико могућности обраде захтева. За почетак ћемо сматрати да постоји сервер који шаље захтев оцењивачу да оцени задатак.

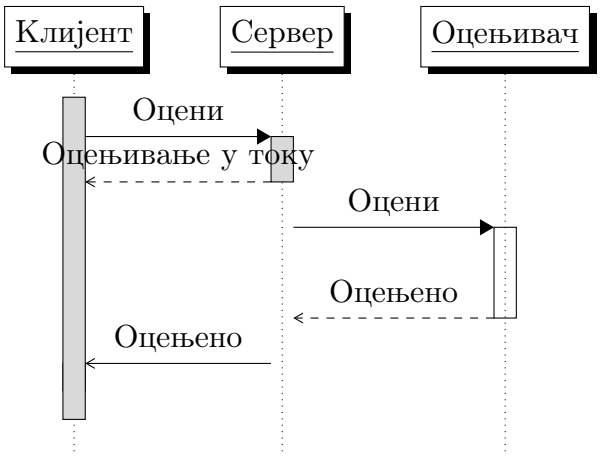
Када оцењивач прими захтев он започиње оцењивање у новом процесу, док се серверу одмах враћа одговор да је задатак у процесу оцењивања и идентификатор оцењивања. Када процес заврши оцењивање он шаље нови

захтев серверу да је оцењивање завршено. Тај захтев треба да садржи идентификатор оцењивања и резултат, односно да ли је задатак успешно решен. Проблем са овим приступом је што оцењивач мора да шаље додатан захтев након што се процес заврши. Предност овог приступа је што нит сервера не би била заузета док се задатак оцењује. Наиме, сервер прима захтев клијента да се оцени задатак, обрађује тај захтев и у одговарајућем формату шаље захтев оцењивачу који врши оцењивање. Нит сервера затим чека одговор оцењивача, у овом случају ће оцењивач одговорити одмах да је оцењивање у току, чиме се ослобађа нит сервера. Сервер мора бити доступан када оцењивач пошаље захтев којим говори да је оцењивање завршено, у супротном се резултат оцењивања губи. На дијаграму 5.2a је приказан овај приступ.

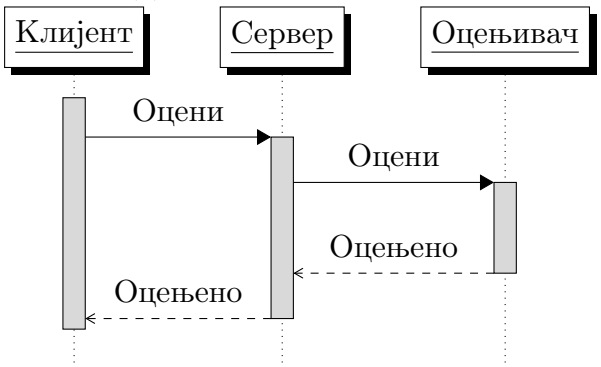
Другим приступом оцењивач не враћа одмах одговор да је оцењивање у току, већ шаље одговор тек када се оцењивање заврши. Овим приступом оцењивач не мора да шаље додатни захтев серверу, међутим то значи да је нит сервера блокирана док се не заврши оцењивање. Добра страна овог приступа је што оцењивач не мора да шаље додатан захтев када заврши оцењивање. До ове тачке оба приступа имају предности и мане, али постоји још један фактор који треба узети у обзир, а то је клијент. Корисник ће из веб прегледача клијентском апликацијом послати захтев за оцењивање серверу, сервер ће обрадити захтев, забележити у базу шта треба и проследити захтев оцењивачу. У претходном приступу ће сервер знати чији одговор је добио на основу одређеног идентификатора. Проблем је што ће сервер морати да води додатну евиденцију да би знао ком клијенту да проследи одговор оцењивача. Незгодно би било ослонити се да ће корисник освежавати страницу и чекати да се добије одговор оцењивача, да не помињемо да ће се тиме повећати број захтева који ће сервер примати. Из тих разлога је овај приступ одабран у имплементацији. На дијаграму 5.2b је приказан овај приступ.

5.5.3 Оцењивање веће количине упита одједном

Корисници ће радити један по један задатак, па претходни захтев за оцењивање има смисла. Али можемо претпоставити да ће постојати ситуација када ће бити погодно послати више захтева на оцењивање одједном. На пример, уколико оцењивач није активан неко време, а задаци послати на оцењивање се бележе у базу. Када се оцењивач поново активира треба оценити задатке који су у међувремену забележени у бази. Администратор тада треба



(a) Први приступ



(b) Други приступ

Слика 5.2: Приступи за оцењивање

да има могућност да пошаље све на оцењивање одједном. Или, на пример, наставник треба да има могућност да пошаље више студентских радова на оцењивање.

Из тих разлога је имплементирана и функција која може да прими захтев за оцењивање више задатака. Захтев је потребно послати POST методом на */grader/checkSolutions* са телом у наредном формату:

```
[
    {
        "requestId": "...",
        "taskId": "...",
        "solution": "...",
        "ordering": "..."
```

```
    }
]
```

Тело захтева се састоји из низа објеката идентичних оним за оцењивање једног задатка у 5.5.2.

За овакво оцењивање је коришћена *Java ExecutorService*. Овим је омогућено креирање групе нити које раде на оцењивању, када једна нит заврши са задатком она преузима следећи. Резултати оцењивања се скупљају у низ који ће бити враћен као одговор серверу. Одговор ће бити у наредном формату:

```
{
"totalTime": "...",
"results": [
        {
        "requestId": "...",
        "status": false,
        "message": "...."
        }
  ]
}
```

Сами одговори у низу *results* су идентични формату који се добије као резултат оцењивања једног задатка у поглављу 5.5.2. Овде се у одговору добија и укупно време извршавања свих упита, односно *totalTime*.

5.5.4 Постављање ограничења

Број захтева које контролер оцењивача може да обради паралелно утиче на даље функционисање система. Наиме, уколико оцењивач користи, на пример, 8 нити за обраду захтева неће се извршавати више од 8 упита паралелно. Ово ограничење иде у прилог и меморијској ефикасности система јер број нити ограничава број корисничких фајлова који постоје у једном тренутку. Експериментално је примећено да величина једног фајла са резултатом не прелази 50 мегабајта што смањује шансу за проблеме са меморијом.

Анализом документације и најбољих пракси установљено је да ће број захтева који се паралелно обрађују одговарати броју поцесора [15]. Оцењивач у позадини има низ захтева које треба обрадити, док се паралелно обрађује само одређени број.

Број захтева може да се промени према потреби, и то на следећи начин:

```
1        Main.threadPoolConfig =
         ↪  transport.getWorkerThreadPoolConfig();
2        threadPoolConfig.setCorePoolSize(threadNo);
3        threadPoolConfig.setMaxPoolSize(threadNo);
4        threadPoolConfig.setQueueLimit(x);
```

У претходном, *threadPoolConfig* представља објекат класе *TCPNIOTransport*, која је повезана са ослушкивачем који реагује на захтеве послате серверу.

Вредности које су постављене ограничавају број захтева који могу да се обрађују у једном тренутку, односно ограничава се број нити које обрађују захтеве. Вредност *setCorePoolSize* задаје иницијалну величину скупа нити, док *setMaxPoolSize* дефинише који је максималан број нити у том скупу. Библиотека има свој ред захтева на чекању. Величина тог реда може да се ограничи и у том случају ће се захтеви који стижу у пун ред одбијати. Максимална величина реда се поставља са *setQueueLimit(x)*. Уколико је прослеђена вредност -1 онда је величина максимална могућа. Уколико се ред напуни нови захтеви ће бити одбијани, односно корисник ће добити поруку да је конекција одбијена.

Максималан број нити који се користи није фиксиран, већ се може проследити као аргумент при покретању програма. Уколико се вредност не проследи подразумевано ће бити 8. Препорука је пре покретања оцењивача извршити тест оптерећења да би се утврдиле оптималне вредности. Такође, препорука је за број нити ставити мању вредност од максималне јер систем потенцијално неће бити оптерећен само оцењивањем већ и другим активностима.

Овде је размотрена могућност коришћења произвољног реда захтева уместо уграђених могућности библиотеке. Било би потребно клијенту одмах вратити одговор да је оцењивач примио задатак. Оцењивач би затим податке из захтева сместио у одговарајућу структуру и у ред. Ред задатака би морао да буде безбедан за рад са нитима. Постојао би скуп нити који узимају задатке

из реда и обрађују их, односно оцењују задатак по задатак. Када се заврши оцењивање шаље се захтев серверу да би га обавестили о резултату задатка. У структури која чува податке захтева би било потребно чување адресе на коју треба слати захтев о завршеном оцењивању. Ово заправо представља дискусију о приступу оцењивању задатака из дела 5.5.2. Као што је ту већ наглашено, одлучено је да се одговор враћа серверу тек када се оцењивање заврши и на тај начин избећи вођење рачуна о томе где треба послати одговор.

5.6 Одређивање оптималних ограничења за извршавање упита

Подразумевамо да је иницијална имплементација у могућности да обради више захтева, али није јасно наведено која су ограничења. У овом поглављу ће бити изложено на који начин су одређена ограничења ресурса за извршавање упита.

Тестирање је извршено на рачунару са процесором *Intel Core i5-8350U* који има 4 језгра и 8 нити, брзине 1.70GHz, и рам меморијом од 16GB. За ово тестирање је имплементирана мала *Java* апликација *LoadTesting* која је паралелно слала захтеве за оцењивање. У циљу тестирања је број нити са којима контролер оцењивача обрађује пристигле захтеве постављен на 8, што одговара броју процесора. Пажљиво је изабрано 20 упита који су слати оцењивачу у циљу тестирања. Од 20 упита 14 чине упити различитих нивоа ефикасности за које се очекује да ће бити успешно оцењени и неће прећи временско ограничење. Преосталих 6 упита је један поновљен неефикасан упит, дат је у наставку:

```
1          SELECT *
2          FROM DA.DOSIJE, DA.ISPIT
```

Претходно наведени упит се извршава преко једног минута уколико нема никаквог ограничења. Приликом ове врсте тестирања је било важно оцењивати и изузетно неефикасне упите да би се проверило да ли један неефикасан упит има утицај на ефикасније упите који се извршавају у исто време. У групи упита за тестирање се нису нашли упити који се не извршавају јер њи-

хово извршавање траје веома кратко и може дати нереалне резултате. Овај тест је имао за циљ да да што боље границе за спорије и упите просечне брзине извршавања.

Први део овог тестирања је био сачињен од паралелног слања иницијалних 20 упита на оцењивање. Циљ овог теста је био одредити параметре за скрипту која прекида неефикасне упите, а која је описана у делу 5.4.3. Параметри које треба одредити су: *ELAPSED_TIME_SEC*, *TOTAL_CPU_TIME* и *QUERY_COST_ESTIMATE*. За почетне вредности су изабране вредности које су добијене након неколико секунди извршавања неефикасног упита, а то су:

- *ELAPSED_TIME_SEC*: 5,
- *TOTAL_CPU_TIME*: 30 000,
- *QUERY_COST_ESTIMATE*: 100 000.

Почетне вредности су дале доста добре резултате. Оцењивач је у више покретања давао 20/20 очекиваних резултата, односно 14 успешно прегледаних задатака и 6 прекорачења. Међутим, на сваких неколико покретања теста су се добијала по један или два упита који прелазе ограничења, а не би требало. Иако је очекивано да количина ресурса коришћених при извршавању једног упита варира, не би требало да постоје превелика одступања.

Након прекидања упита није била доступна информација о томе које од три задата ограничења је било прекорачено. Да би се видело које ограничење је упит достигао, скрипта за постављање ограничења је проширена да исписује вредности параметара за упит који је испунио услове за прекидање. Примећено је да су упити прекидани због временског ограничења. Извршавањем мање ефикасних упита је примећено да је њихово време извршавања било између 4 и 8 секунди, па је временско ограничење повећано на 10 секунди.

Поновљеним тестом је примећено да се и даље, с времена на време, исти упити неуспешно оцењују. Али сада није прекорачено време, већ параметар *QUERY_COST_ESTIMATE*, па је та вредност била постепено повећавана уз поновно тестирање. Скрипта је пре прекидања упита исписивала коју вредност овог параметра је упит имао, па је у складу с тим вредност ограничења повећавана. На крају тестирања се вредност 400 000 показала као оптимална.

Исти упит може заузети различите количине ресурса од извршавања до извршавања, па је овде циљ наћи оптималну границу за сваки ресурс.

Током тестирања упити нису били прекидани због ограничене вредности *TOTAL_ CPU_ TIME*. Када су упити били прекидани због друга два ограничења, приказиване су и вредности процесорског времена које су биле утрошене. Како су те вредности биле мало испод задатих 30 000, одлучено је да није потребно смањивати вредност овог ограничења и да је иницијална вредност оптимална.

Ограничења су проверена са 10 понављања теста. Један тест је подразумевао слање 20 упита на оцењивање. На крају је закључено да су наредни параметри дали добре резултате, односно да су успешно оцењени сви упити за које се то очекује:

- *ELAPSED_ TIME_ SEC*: 10,
- *TOTAL_ CPU_ TIME*: 30 000,
- *QUERY_ COST_ ESTIMATE*: 400 000.

Када су одређена ова ограничења проверено је да ли оцењивач може успешно да прегледа више радова. Паралелно је слато 20, 50 и 100 упита на оцењивање, и сваки од случајева је поновљен 10 пута. Свако извршавање је дало очекиване резултате.

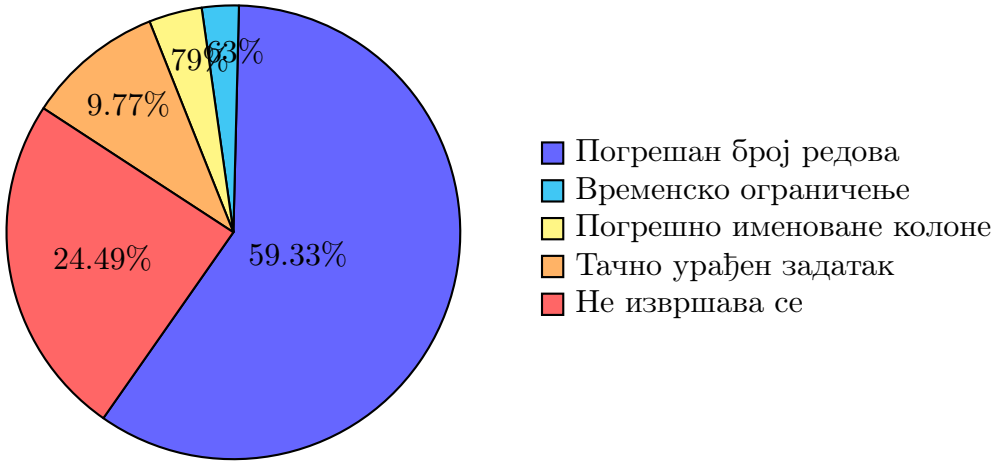
Приликом тестирања оптерећења имати у виду да систем *Db2* може паралелно да извршава само одређени број упита, у зависности од могућности система. Уколико систем добије више упита него што може да изврши он ће их сместити у ред. Потребно је проценити да ли ће се користити већи број нити за обраду захтева и тиме смештати више захтева у ред система *Db2*, или ће се користити мање нити, чиме ће се ослободити *Db2*, док ће ред захтева чувати *Java* апликација. У овом случају је одлучено да се користи мање нити и смањи оптерећење СУБП. Разлог за ову одлуку је што је приликом тестирања примећено да перформансе система опадају са повећањем броја нити, односно броја упита који се истовремено извршавају. Односно, неки упити који није требало да буду прекинути су прекорачили претходно задата ограничења. Ово даље имплицира да са повећањем број нити треба поново одредити ограничења.

5.7 Тестирање исправности оцењивача

Оцењивач не сме дати одговор да је задатак тачан уколико није, и не сме дати одговор да задатак није тачан уколико јесте. У овом поглављу ће бити представљено тестирање које је вршено над оцењивачем.

За тестирање су коришћени студентски радови са испита из предмета *Релационе базе података* школске 2022/2023 и дела 2023/2024. У анализама задатка се не налазе информације о студентима, већ само упити које су студенти писали. Тестирање се врши слањем захтева за тестирање групе радове. Оцењивач обрађује захтеве као што је описано у поглављу 5.5.3. Оцењивач користи скуп од 8 нити које су оцењивале задатке.

Тестирано је 14 различитих задатака. Тестирање је вршено над 643 студентска рада. Упити су слати на оцењивање у групама. Једну групу чине сви радови за један задатак, групе су имале између 21 и 75 радова. Сваки одговор оцењивача је проверен од стране аутора. У 43 студентска рада су постојале мање грешке, попут словних, погрешног формата исписа, именовања колона и слично. Ти радови су исправљени и оцењени још једном. Узевши у обзир оригиналне и исправљене радове оцењивач је укупно оценио 686 радова. Расподела одговора оцењивача је приказана на дијаграму 5.3.



Слика 5.3: Резултати добијени од оцењивача

Сви резултати су били у складу са очекивањима, али је обраћена посебна пажња на следеће:

- Уколико студент дода сортирање које се није тражило у задатку добиће одговор да се упит не извршава. Наиме, уколико се у задатку не тражи

сортирање оцењивач ће га сам додати. Ако је студент додао *order by*, а оцењивач га дода још једном проузроковаће синтаксну грешку и упит се неће извршити. Иако порука не даје тачан узрок проблема, строго гледано задатак није тачно урађен и тај одговор је очекиван.

- Друга верзија претходне грешке је да се у задатку тражило сортирање а студент га није додао. Тачније, уколико сортирање у задатку није довољно да одреди јединствен редослед, део сортирања ће бити додат од стране оцењивача. Очекивано је ће студентски рад на крају имати *order by x, y*, а да ће оцењивач додати *,2,3* на пример. Уколико студент није додао сортирање додавање ће само додати запету и низ бројева, што је довело до тога да се упит не извршава због синтаксне грешке. Стриктно гледано, студентско решење није тачно, али порука неће јасно указати студенту шта је проблем.

- Уколико студент не користи одговарајућу схему добиће поруку да се упит не извршава. Корисник на оцењивање шаље један упит, према томе он не може променити подразумевану схему. Подразумевана схема је *student*, док се табеле у овој конкретној имплементацији налазе у схеми *da*. Према томе, студент табели испита треба да приступа са *da.ispit*, а не само са *ispit*. Променом скрипте је могуће поставити подразумевану схему, али у конкретној имплементацији је одлучено да се то не ради да би студенти увежбали коришћење схеме. У овом случају очекивано да се упит не извршава.

- У свега пар примера је забележено да су студенти правили додатне табеле или функције. Напоменимо још да је коришћење *WITH* клаузе за прављење помоћних табела дозвољено, али прављење физичке табеле са *CREATE TABLE* није. Решење задатка треба да буде тачно један упит. У случајевима када су студенти правили додатне функције и табеле су добили одговор да се упит не извршава, прави разлог томе је што немају неопходна права.

- Именовање колона се јавило као проблем, а то је већ разматрано у делу 5.1. Наиме, за тестирање су коришћени испитни задаци у којима се није увек тражило именовање колона. Уколико се име колоне није тражило, очекивано је да то буде редни број. Уколико је студент именовао ко-

лону, његов резултат неће бити исти као тачан резултат. Касније је у примерима примећено да уколико студент користи колону из помоћне табеле, та колона мора имати име, односно неће бити редни број. Овај проблем је решен тиме што су задаци измењени тако да се у сваком захтева именовање колона.

- Друге најчешће грешке су биле грешке у испису. На пример, ако у задатку стоји да колона треба да садржи испис наредног формата *<ime studenta> je student.* Очекивано је да ће *<ime studenta>* бити замењено именом студента, али студенти су неретко остављали карактере < и >. Очекиван испис је, на пример, *Milica je student*, а добијано је *<Milica> je student.* Други пример би био да су неки студенти стављали тачку на крај исписа, а неки не. Још једна честа грешка је била да студент стави *ž, ć, č* уместо *ошишане латинице*, *z, c.* То је наравно нешто на шта се не обраћа велика пажња на испиту. Али што се оцењивача тиче, испис треба да буде тачан у потпуности, односно ако се не тражи тачка онда и не треба да је буде. Овде је важно нагласити да задаци за оцењивач треба да буду јасно дефинисани да не би било додатне забуне.

- Било је и примера стандардних синтаксних грешака, попут погрешно наведене структуре *SQL* наредбе, коришћење погрешне врсте наводника, запете на местима где не треба да буду.

Напоменимо још једном, поруке о грешци се прослеђују из скрипте. Према потреби се скрипта јако лако може модификовати да даје дескриптивније поруке, на пример да је проблем са сортирањем или схемом, на основу променљиве *SQLSTATE* која се добија при извршавању упита у поруци о грешци.

Глава 6

Имплементација сервера

Сервер представља програм који је задужен за одржавање базе са корисницима, задацима, као и комуникацију између клијентске апликације и одговарајућих оцењивача. У овом делу ће прво бити представљена структура базе сервера, а затим и све његове могућности.

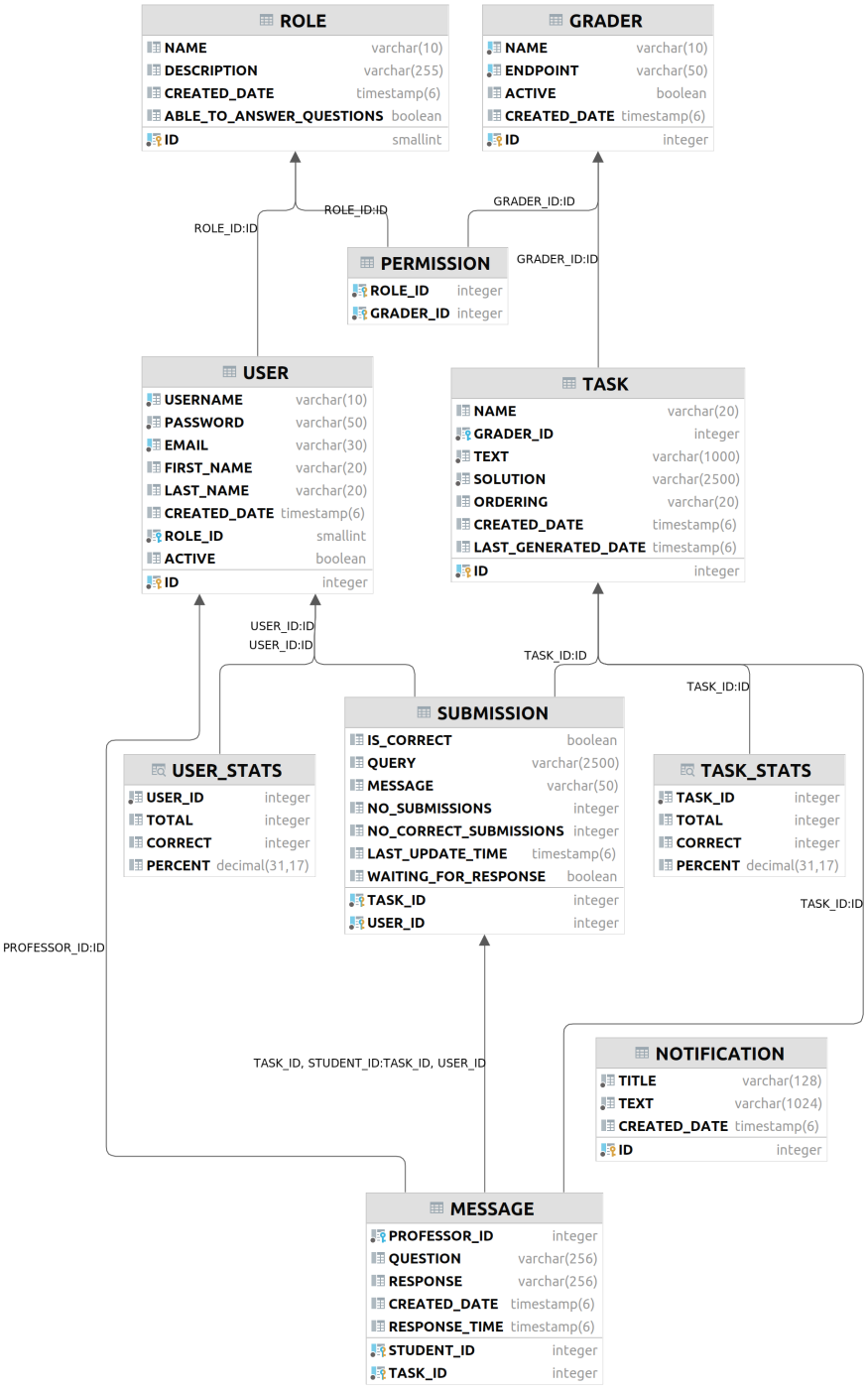
6.1 Серверска база

Серверска база је названа *grader* и њена структура је приказана на дијаграму 6.1. У наставку ће бити објашњена улога сваке табеле у бази.

6.1.1 Табела Role

Ова табела представља улоге корисника у систему. На основу улоге корисника се одређује колика права он има, као и којим задацима има приступ. Инцијално су замишљене 4 улоге:

- *администратор* - корисник са највећим привилегијама, има директан приступ бази и право да додељује више улоге корисницима;
- *професор* - корисник који је задужен за задатке који се оцењују. Има право да додаје и мења задатке преко интерфејса, да одговара студентима на питања као и да поставља обавештења;
- *студент* - представља студента Математичког факултета. Он има приступ дефинисан табелом приступа која ће бити представљена у наставку.



Слика 6.1: Схема базе *grader*

Кориснике са овом улогом карактеришу корисничко име и мејл адреса тачно одређеног формата одређеног од стране факултета;

- *остали* - корисници система који не спадају ни у једну од претходних

категорија. Они немају приступ бази са којом раде студенти факултета, али могу имати приступ неком другом потенцијалном оцењивачу.

У табели се налазе наредне колоне са одговарајућим значењем:

- *ID* - јединствени идентификатор улоге. Представља примарни кључ и вредност је аутоматски генерисана;
- *NAME* - текст који представља назив улоге;
- *DESCRIPTION* - текст који представља опис улоге;
- *CREATED_DATE* - време када је улога креирана;
- *ABLE_TO_ANSWER_QUESTIONS* - индикатор који даје информацију да ли корисник са датом улогом има право да одговара на студентска питања.

Администратор има могућност да дода потпуно нову улогу. На пример, улогу за ученике средњих школа. Уз додавање нове улоге ће морати да се дода и право приступа које улога има, а то ће бити описано нешто касније.

6.1.2 Табела User

Табела *User* је намењена чувању корисничких налога. Свако ко жели било коју врсту приступа систему мора да има налог.

У табели се налазе наредне колоне са одговарајућим значењем:

- *ID* - јединствени идентификатор корисника. Представља примарни кључ и вредност је аутоматски генерисана.
- *USERNAME* - корисничко име. Мора бити јединствено и користиће се за логовање. За одређене улоге корисничко име ће морати да има одговарајући формат.
- *PASSWORD* - шифра кодирана мд5 алгоритмом.
- *EMAIL* - мејл адреса корисника ће морати да буде јединствена.
- *FIRST_NAME* - име корисника није обавезно.

- *LAST_NAME* - презиме корисника није обавезно.
- *CREATED_DATE* - време када је корисник креиран.
- *ROLE_ID* - страни кључ ка табели улога. На основу њега ће корисник добити одговарајућа права приступа.
- *ACTIVE* - индикатор да ли је кориснички налог активан. Идеја је да се после извесног времена некоришћења налога исти деактивира.

Како корисничко име мора бити јединствено размотрена је могућност да оно буде примарни кључ. Одлучено је да то ипак буде нова аутоматски генерисана вредност.

Постоје и други новији алгоритми за енкодирање, попут *Argon2* и *Bcrypt*. Ова два алгоритма врше енкодирање знатно спорије од алгоритма *мд5* и то их чини безбеднијим. Уколико је алгоритам енкодирања брз, грубом силом се у кратком временском року може пробати више могућности и на тај начин открити шифра. У неком будућем унапређењу система се може прећи на новије алгоритме.

Додела улоге кориснику

У захтеву за креирање налога корисник може да нагласи коју улогу жели, али то не мора да значи да ће је и добити. Подразумевана улога је *остали*.

Постојало је неколико идеја за доделу улога. Прва је била да се у коду дода неколико условних наредби које би доделиле улогу. Овим би свако додавање улога, или пак промена захтева за улогу, водило променама у коду, заустављању сервера и покретању његове нове верзије. То не делује као велики проблем, али боље би било да сервер не мора да се зауставља за овако нешто.

Друга идеја је била да се у табели улога чувају и услови које кориснички налог треба да испуни да би имао ту улогу. На пример, да се чува колона са регуларним изразом који би требало да испуне мејл адреса или корисничко име. Овим би се измена услова свела на промену вредности у табели, а то би могао администратор да уради без измена у коду. То би ограничило услове на оно што могу да ураде регуларни изрази, што није довољно флексибилно. На пример, уколико желимо да је корисничко име почетак мејл адресе, потребно

је искористити два поља. Да ли онда додати колону за ту комбинацију? Односно, да ли би за услов који користи колону мејл и колону корисничко име било потребно креирати нову колону са условом. Ово би додатно закомпликовало услове.

За решење овог проблема је потребно нешто што даје могућности кода, али да није код сервер који захтева рекомпилацију при свакој измени. Оно што даје могућност кода, а није код су окидачи (енг. *trigger*) у бази података. Измена окидача не захтева заустављање сервера, а могу се направити компликованији услови. У бази се налази окидач *SET_USER_ROLE* који решава управо овај проблем. Окидач ће се извршавати пре уноса корисника, али и пре његове измене. Уколико корисник промени своју мејл адресу тако да више не испуњава услове он треба да изгуби улогу која има. Окидач ће бити приказан у делу 6.1.10.

6.1.3 Табела Grader

Ова табела је намењена чувању доступних оцењивача. Сервер има могућност да подржи више оцењивача. Додавање оцењивача је могуће без икаквих измена у имплементацији сервера, већ само додавањем одговарајућих вредности у ову табелу. Поступак за додавање оцењивача ће бити објашњен у делу 6.2.

Идеја је да сваки оцењивач има своју базу која се користи за оцењивање задатака. База *stud2020* се налази у оцењивачу под називом *ocenjivac_stud2020*, док би се, на пример, база мастер студената налазила у оцењивачу *ocenjivac_mstud2024*.

Иницијално се у овој табели налази само оцењивач који користи за базу *stud2020* чија је имплементација објашњена у претходном делу.

У табели се налазе наредне колоне са одговарајућим значењем:

- *ID* - јединствени идентификатор оцењивача. Представља примарни кључ и његова вредност је аутоматски генерисана.
- *NAME* - назив оцењивача. Ова информација се приказује кориснику.
- *ENDPOINT* - текст који представља адресу на којој се налази оцењивач, односно где треба слати захтев за оцењивање.
- *CREATED_DATE* - време када је оцењивач креиран.

- *ACTIVE* - индикатор да ли је оцењивач активан. Уколико оцењивач није активан корисници неће имати приступ задацима за које је задужен тај оцењивач.

6.1.4 Табела Permission

Табела *Permission* се користи за одређивање којим оцењивачима корисник има приступ. Ову табелу ћемо звати још и табела права приступа оцењивачу. Као што је помињано раније, сервер може да се прошири новим оцењивачем без измена кода. Ова табела омогућава давање приступа одговарајућим улогама за одговарајући оцењивач. Рецимо ако би додали улогу за ученике средњих школа, они не треба да имају приступ бази која се користи за часове на факултету, већ бази прилагођеној њиховом плану и програму.

Ова табела има само наредне две колоне чија комбинација уједно чини и примарни кључ:

- *ROLE_ID* - страни кључ ка табели улога;
- *GRADER_ID* - страни кључ ка табели оцењивача.

Када додамо нову улогу, потребно је и у табелу са правима приступа додати ком оцењивачу корисник са новом улогом има приступ. За студенте и остале кориснике би то представљало које задатке могу да раде. За наставнике би то значило којим оцењивачима могу да додају задатке и за које оцењиваче су одговорни. На пример, уколико имамо оцењивач ком наставници немају приступ, њему ни један наставник неће моћи да дода задатке. Администратор треба да има приступ свим оцењивачима.

Правило важи и са друге стране, односно при додавању новог оцењивача, потребно је додати приступ том оцењивачу. Односно који корисници ће моћи да раде задатке на том новом оцењивачу и да ли ће група наставника моћи да му додаје задатке.

6.1.5 Табела Task

Табела *Task* је намењена чувању свих задатака, без обзира на то који оцењивач је задужен за њихово оцењивање.

У табели се налазе наредне колоне са одговарајућим значењем:

- *ID* - јединствени идентификатор задатка. Представља примарни кључ и вредност је аутоматски генерисана.
- *NAME* - текст који представља назив задатка. Може бити нешто карактеристично за задатак или, на пример, ако је задатак са испита, у ком је року био.
- *GRADER_ID* - страни кључ ка табели оцењивача који је задужен за оцењивање задатка.
- *TEXT* - текст са формулацијом задатка.
- *SOLUTION* - текст са упитом који представља тачно решење задатка.
- *ORDERING* - текст са сортирањем које се додаје на упит, може бити на пример **1,2,3**, или **,2,3**, као што је појашњено у делу 5. Ово сортирање се додаје и на тачно решење и на студентске покушаје.
- *ACTIVE* - индикатор да ли је задатак активан, односно доступан за решавање.
- *CREATED_DATE* - време када је задатак креиран.
- *LAST_GENERATED_DATE* - време када је тачан резултат био генерисан на оцењивачу задњи пут. Регенерисање резултата се врши слањем POST захтева оцењивачу на путању */grader/generateResults* као што је појашњено у делу 5.4.1.

Додавање задатака ће детаљно бити размотрено у делу 6.2.6, а сада ће укратко бити размотрено брисање задатака. Сваки оцењивач има индикатор који сигнализира да ли је активан, а самим тим да ли задаци за које је он задужен могу да се раде. Уколико бришемо задатак треба имати у виду да ли је неки корисник већ радио тај задатак, јер ако јесте онда би се и његов резултат изгубио. Теоретски, информација о корисничком резултату би могла да остане, али не би имало пуно смисла ако обришемо задатак са његовим текстом. Због овога је одлучено да се задаци не бришу, већ само деактивирају уколико не треба више да буду доступни корисницима. Дакле, корисницима су доступни активни задаци за чије оцењивање је задужен активан оцењивач.

Сви задаци могу да се мењају. При измени је само важно уверити се да је на оцењивачу регенерисан резултат, а то се може видети у одговору захтева за регенерацију.

6.1.6 Табела Submission

Ред ове табеле представља један захтев за оцењивање задатка, па ћемо је звати још табела захтева. Сваки корисник ће за сваки задатак који је покушао да реши имати по један ред у овој табели. Уколико корисник исти задатак жели да решава више пута неће се додавати нови редови, већ ће се мењати постојећи. Када корисник покуша поново да реши задатак његов претходни покушај ће бити изгубљен. Ова одлука је донета у циљу меморијске оптимизације. Узевши у обзир да цео систем који овај рад представља може обухватити више оцењивача јасно се види да је потребно водити рачуна о меморији коју исти заузима.

Наредне колоне се налазе у овој табели:

- *TASK_ID* - јединствени идентификатор задатка који је решаван. Представља страни кључ ка табели задатака и део је примарног кључа.
- *USER_ID* - јединствени идентификатор корисника који решава задатак. Представља страни кључ ка табели корисника и део је примарног кључа.
- *QUERY* - текст који прдставља упит који корисник шаље на оцењивање.
- *IS_CORRECT* - индикатор да ли је корисник тачно решио задатак задњи пут када је оцењен.
- *MESSAGE* - порука коју оцењивач враћа након оцењивања. Уколико је задатак тачно решен то ће бити време извршавања, у супротном ће бити нека од индикација о грешци.
- *NUM_SUBMISSIONS* - број колико пута је корисник слао захтев за оцењивање овог задатка.
- *NUM_CORRECT_SUBMISSIONS* - број колико пута је корисник тачно решио задатак.

- *LAST_UPDATE_TIME* - време задње модификације реда, може бити време када је оцењивач одговорио, или време када је задатак послат на оцењивање.

- *WAITING_FOR_RESPONSE* - индикатор да ли се чека на одговор оцењивача, корисник не треба да има могућност да шаље задатак на оцењивање други пут уколико још увек није добио одговор за први покушај.

Уколико је корисник погрешно решио задатак имаће могућност да покуша поново. Али и када корисник тачно реши задатак неће му се одузети могућност да покуша поново да га реши. Разлог томе је што корисник може да се сети неког другачијег приступа или ефикаснијег решења након што је већ послао једно тачно решење, па нема смисла одузимати му могућност да и друго решење тестира.

Индикатор да ли се чека на одговор оцењивача има још једну улогу. Наиме, уколико се оцењивач искључи из неког разлога, а захтеви и даље стижу ти захтеви ће бити забележени у табели и остаће у стању чекања. Када се оцењивач поново покрене наставник и администратор имају могућност да на клик дугмета пошаљу све захтеве који су заглавили у стању чекања на поновно оцењивање. Напоменимо још да ова недоступност оцењивача подразумева да је оцењивач у табели оцењивача и даље активан и да због тога корисници и даље имају приступ задацима.

6.1.7 Табела Message

Табела порука омогућава кориснику да постави питање наставнику уколико има нејасноћа са решењем које је послао. Када корисник добије одговор оцењивача о његовом решењу добиће могућност да кликом на дугме пошаље питање наставнику.

У овој табели се налазе наредне колоне:

- *STUDENT_ID* - идентификатор корисника који поставља питање. Део је примарног кључа и део страног кључа ка табели захтева.

- *TASK_ID* - идентификатор задатка за који се поставља питање. Део је примарног кључа и део страног кључа ка табели захтева.

- *PROFESSOR_ID* - идентификатор наставника коме је постављено питање. Страни кључ ка табели корисника.
- *QUESTION* - текст питања које поставља корисник.
- *RESPONSE* - одговор наставника.
- *CREATED_DATE* - време креирања поруке, односно када је корисник поставио питање.
- *RESPONSE_DATE* - време када је наставник одговорио на питање.

Да би се наставнику могло поставити питање, за њега морају бити испуњени наредни услови:

1. Мора да има улогу са могућношћу да одговара на питања, односно индикатор *ABLE_TO_ANSWER_QUESTIONS* његове улоге треба бити постављен на тачно.
2. Мора имати приступ оцењивачу за који се поставља питање. То је одређено табелом са правима приступа.
3. Кориснички налог мора бити активан. Уколико наставник оде на одсуство може привремено да тражи да му се деактивира налог да му корисници не би постављали питања.

Корисник има могућност да постави једно питање по задатку. Уколико би желели више питања то би подразумевало имплементацију целе конверзације, што није потребно у датом систему. Предлог је да уколико корисник има више недоумица контактира наставника путем мејла и по потреби закаже консултације.

Након што корисник постави питање постоји више могућности за даљи ток. Једна могућност је да настави да ради исти задатак. У том случају са питањем треба сачувати и кориснички упит за који се поставља питање. Друга идеја је забранити кориснику да решава задатак док не добије одговор. Јер уколико је корисник поставио питање сматра се да му нешто око задатка није јасно и да не може више да покушава. У овом приступу се јавља проблем да се корисник можда сети како да реши задатак, али већ је поставио питање па не може да испроба ново решење. Да би се и ово решило, корисник има

могућност да обрише своје питање пре него што наставник одговори. Тиме ће добити могућност да поново ради задатак. Уколико је наставник одговорио корисник ће имати могућност да поново ради задатак, али неће моћи да обрише поруку са одговором наставника.

Одлучено је да се имплементација система за постављање питања додатно не проширује.

6.1.8 Табела Notification

Табела *Notification* се користи за чување обавештења. С времена на време ће бити новости на серверу, на пример о додавању нових задатака. Сва та обавештења је потребно чувати у овој табели.

Ова табела ће имати наредне колоне:

- *ID* - примарни кључ који ће бити аутоматски генерисан.
- *TITLE* - наслов обавештења.
- *TEXT* - текст обавештења. Може бити обичан текст или *HTML*.
- *CREATED_DATE* - датум и време када је обавештење креирано.

Размотрено је прављење додатне базе која би била коришћена само од стране клијентског дела, али се то показало као непотребно јер би била потребна само једна табела.

Обавештења могу да постављају администратори и наставници.

6.1.9 Погледи серверске базе

У бази се налазе два наизглед идентична погледа, *USER_STATS* и *TASK_STATS*. Оба погледа су намењена ранг листама, први представља успешност корисника, док други представља колико добро је урађен сваки задатак. Оба погледа врше израчунавање на основу табеле са захтевима за оцењивање задатка (*Submissions*).

Поглед *USER_STATS* има наредне колоне:

- *USER_ID* - јединствени идентификатор корисника из табеле корисника;
- *TOTAL_ATTEMPTS* - укупан број покушаја корисника да се реши задатак. Овде се броје и поновљени покушаји за исти задатак;

- *CORRECT_ATTEMPTS* - укупан број успешно оцењених задатака. Овде се пребројава и ако је корисник више пута тачно решио задатак;
- *PERCENT_OF_SOLVED_TASKS* - процентуални удео успешно решених задатака у свим решаваним задацима. Овде се не броји колико пута је корисник покушао да реши задатак, већ да има једно успешно решење.

Поглед се креира нередним упитом:

```
1 CREATE OR REPLACE VIEW USER_STATS AS
2 SELECT USER_ID, SUM(NO_SUBMISSIONS) TOTAL_ATTEMPTS,
  ↪  SUM(NO_CORRECT_SUBMISSIONS) CORRECT_ATTEMPTS,
3 SUM(CASE WHEN NO_CORRECT_SUBMISSIONS > 0 THEN 1 ELSE 0 END) *
  ↪  100.0 / COUNT(*) PERCENT_OF_SOLVED_TASKS
4 FROM SUBMISSION
5 GROUP BY USER_ID;
```

Поглед *TASK_STATS* има наредне колоне:

- *TASK_ID* - јединствени идентификатор задатка из табеле задатака;
- *TOTAL_ATTEMPTS* - укупан број покушаја да се реши овај задатак. Овде се броје и поновни покушаји;
- *CORRECT_ATTEMPTS* - укупан број успешно оцењених решења овог задатка. Броји се и ако је један корисник више пута успешно решио задатак;
- *PERCENT_OF_SOLVED_TASKS* - процентуални удео успешно оцењених задатака у свим оцењеним задацима. Овде се не броје поновни покушаји.

Поглед се креира нередним упитом:

```
1 CREATE OR REPLACE VIEW TASK_STATS AS
2 SELECT TASK_ID, SUM(NO_SUBMISSIONS) TOTAL_ATTEMPTS,
  ↪  SUM(NO_CORRECT_SUBMISSIONS) CORRECT_ATTEMPTS,
```

```
3  SUM(CASE WHEN NO_CORRECT_SUBMISSIONS > 0 THEN 1 ELSE 0 END) *
   ↪  100.0 / COUNT(*) PERCENT_OF_SOLVED_TASKS
4  FROM SUBMISSION
5  GROUP BY TASK_ID;
```

6.1.10 Окидачи серверске базе

Серверска база има један окидач над табелом корисника. Овај окидач има за циљ да кориснику пре креирања или измене налога постави одговарајућу улогу. У овом окидачу се улога поставља на основу корисничког имена и мејл адресе, али се услови лако могу проширити.

Команда за креирање окидача је у наставку:

```
1  CREATE OR REPLACE TRIGGER SET_USER_ROLE
2  BEFORE INSERT OR UPDATE
3  ON USER
4  REFERENCING NEW AS N
5  FOR EACH ROW
6  BEGIN
7  SET N.ROLE_ID = CASE
8          WHEN N.EMAIL LIKE 'mi_____@alas.matf.bg.ac.rs'
9                  AND N.USERNAME LIKE 'mi_____'
10                 AND N.USERNAME = SUBSTR(N.EMAIL, 1, 7)
11                 AND LENGTH(EMAIL)>LENGTH(USERNAME)
12         THEN 3
13         WHEN N.EMAIL LIKE 'mr_____@alas.matf.bg.ac.rs'
14                 AND N.USERNAME LIKE 'mr_____'
15                 AND N.USERNAME = SUBSTR(N.EMAIL, 1, 7)
16                 AND LENGTH(EMAIL)>LENGTH(USERNAME)
17         THEN 3
18         WHEN N.EMAIL LIKE '%@poincare.matf.bg.ac.rs'
19                 AND LENGTH(EMAIL)>LENGTH(USERNAME)
20                 AND SUBSTR(N.EMAIL, 1, LENGTH(N.USERNAME)) =
   ↪  USERNAME
```

```
21          THEN 2
22          ELSE 4 END;
23  END@
```

6.1.11 Индекси серверске базе

Индекси се креирају са циљем бржег приступа подацима. Када приступамо захтевима за оцењивање у највећем броју случајева ће нам требати група захтева једног корисника. На пример, када се корисник логује потребно је дохватити све задатке које је претходно решавао. Табела захтева за оцењивање ће бити индексирана по идентификатору корисника, односно биће креиран кластерован индекс. Можемо замислити да су сви радови једног корисника чувани узастопно у табели, те да ће СУБП знати ефикасно да им приступи. Предност ове оптимизације ће се видети тек са нешто већим бројем корисника.

Индекс је креиран наредним упитом:

```
1  CREATE INDEX SUBMISSION_ORDER_BY_USER
2          ON SUBMISSION (USER_ID ASC);
```

Можемо размотрити индексирање табеле са порукама. Наиме, при логовању корисник треба да добије и питања која је поставио, односно питања на која треба да одговори. Порука се спаја са тачно једним захтевом за оцењивање, па се са њим и шаље. Уколико индексирамо по кориснику који је поставио питање, као у табели захтева, онда би корисник који је поставио питање ефикасно добио одговор. Али ако погледамо са друге стране, односно корисника који треба да одговори на питање онда дохватање питања није тако ефикасно. У том случају би више помогао кластерован индекс по наставницима.

На основу претходног се да закључити да би за табелу порука било погодно кластеровање по кориснику који поставља питање и кориснику који треба да одговори. Уколико при креирању индекса прво наведемо корисника који поставља питање онда ће дохватање питања за њега бити нешто ефикасније.

Ово је свакако побољшање и за једног и за другог корисника. Наредни упит креира овај индекс:

```
1 CREATE INDEX MESSAGE_ORDER_BY_STUDENT_AND_PROFESSOR
2 ON MESSAGE (STUDENT_ID ASC, PROFESSOR_ID ASC);
```

За табеле са задацима и обавештењима сервер има помоћне документе као вид оптимизације и смањења броја приступа бази, као што је поменуто у поглављу 4. Табеле оцењивача, улога и права приступа имају мали број редова и не очекује се да ће се тај број знатно повећати проширивањем система. Према томе индексирање за њих није потребно.

6.2 Имплементација контролера

Као и оцењивач, сервер је имплементиран у језику *Java*, коришћењем библиотеке Grizzly за руковање *HTTP* захтевима.

Како се очекује да ће сервер имати више захтева идеја је омогућити да један захтев одради што више ствари да би се смањила потреба за слањем додатних захтева. Додатно треба узети у обзир да број конекција на базу података, иако може бити велики, и даље је ограничен. Број захтева бази је минимизован ради повећања ефикасности.

Када се сервер први пут покрене неопходно је конфигурисати га. Конфигурација подразумева покретање оцењивача, његово додавање у табелу оцењивача и додавање права приступа унетом оцењивачу. Ово је задужење администратора. Када је сервер покренут постоји неколицина уобичајених активности, попут креирања налога, логовања, решавања задатака, постављања питања, што већином раде корисници са улогом студента. Корисници са улогом наставника имају задужења да додају задатке, да их мењају по потреби, додају обавештења као и да одговарају на питања.

Све ове могућности сервера ће бити описане у наставку.

Покретање оцењивача

Додавање оцењивача подразумева додавање једног сервиса који има могућност да оцењује задатке. Нови оцењивач не мора нужно бити за задатке

из *SQL-a*, већ може бити за произвољан програмски језик. У овом раду ћемо се фокусирати искључиво на *SQL*.

Да би оцењивач био компатибилан са овим сервером мора имати могућност да прими наредна три POST захтева:

1. */grader/generateResults* - за генерисање резултата на основу тачних решења у оцењивачу. Овај захтев припрема оцењивач за оцењивање задатака.

2. */grader/checkSolution* - за проверу исправности једног задатка;

3. */grader/checkSolutions* - за проверу исправности групе задатака.

Формат сваког од ова три захтева је објашњен у делу 5.5 са имплементацијом оцењивача.

Имплементација оцењивача дата у делу 5 може се уз измену искористити за исти или други систем за управљање базом података. У случају *Db2* система са другом базом података потребно је само у скрипти изменити на коју базу се врши повезивање. У случају другог система, потребно је изменити скрипту у већој мери, али апликација за оцењивање може остати идентична. На пример, у случају *PostgreSQL* система ће уместо:

```
db2 "$query" | sed '/^$/d' > "$user_path"
```

бити коришћено нешто попут:

```
psql -c '\x' -c '$query' > "$user_path"
```

Уколико би хтели да се оцене задаци неког императивног програмског језика оцењивач би морао суштински да се промени. Наиме, заштита система од корисничког програма писаног у, на пример, *C-y* се доста разликује од заштита базе од корисника. За имплементацију таквог оцењивача се може реферисати мастер рад Огњена Коцића [19].

Препорука је да се оцењивач покрене унутар докер контејнера и да се за њега узме у обзир све изложено у делу 5.

Додавање оцењивача у табелу оцењивача

Када је оцењивач спреман да прима захтеве потребно је да сервер добије информацију о томе. Први корак у том смеру је његово додавање у табелу оцењивача. Важно је ставити одговарајућу локацију оцењивача (*endpoint*) и да је оцењивач активан.

Додавање и измена оцењивача се могу вршити слањем наредних захтева серверу.

POST захтевом на */grader/addGrader* са наредним телом додаје се оцењивач:

```
{
        "name": "stud2020",
        "endpoint": "http://localhost:51000",
        "active": true
}
```

Додати оцењивач ће имати назив *stud2020*, биће могуће приступити му на адреси *http://localhost:51000* и иницијално је активан.

Уколико је потребно изменити оцењивач то се може урадити слањем идентичног захтева на */grader/updateGrader*. У захтеву за измену треба да се нађе и идентификатор оцењивача:

```
{
        "id": 1,
        "name": "stud2020",
        "endpoint": "http://localhost:51000",
        "active": true
}
```

Овим се може активирати и деактивирати оцењивач, као и променити адреса оцењивача у случају да је то потребно.

Списак активних оцењивача се може добити слањем GET захтева на */grader/getGraders*, док се списак свих оцењивача може добити слањем GET захтева на */grader/getAllGraders*.

Додавање права приступа

У овом тренутку сервер има информацију о новом оцењивачу али то не значи да корисници имају приступ. Да би то било решено потребно је одговарајућим корисницима дати приступ. Како се приступ не даје директно сваком корисника, него улогама, потребно је додати информације у табелу са правима приступа. Корисник ће на основу своје улоге имплицитно добити приступ.

Слањем POST захтева на */roleGraderPermission/addPermissions* са наредним телом ће бити додати одговарајући приступи:

```
[
        {
                "roleId": 1,
                "graderId": 1
        }
]
```

Јединствени идентификатор улоге треба поставити у *roleId*, док *graderId* представља јединствени идентификатор новог оцењивача.

Слањем идентичног захтева на */roleGraderPermission/removePermission* ће бити уклоњена права приступа улоге оцењивачу.

Када се додају права приступа треба размотрити који све корисници треба да га имају. У већини случајева администратор треба да има приступ, као и наставници.

6.2.1 Креирање корисничког налога

Нови корисник се креира слањем POST захтева на */user/create* са наредним телом:

```
{
        "username": "...",
        "email": "...",
        "firstname": "...",
```

```
5           "lastname": "...",
6           "password": "..."
7   }
```

У захтеву се траже наредна поља:

- *username* - корисничко име. Мора бити јединствено.
- *email* - мејл адреса. Мора бити јединствена.
- *firstname* - име. Поље није обавезно.
- *lastname* - презиме. Поље није обавезно.
- *password* - шифра.

Кориснички налог ће бити креиран уколико већ не постоји корисник истог корисничког имена и мејл адресе. Након што је налог успешно креиран аутоматски ће се улоговати. Резултат логовања ће бити описан у наставку.

6.2.2 Логовање

Захтев за логовање се шаље POST захтевом на */user/login* са наредним телом:

```
1   {
2           "username": "...",
3           "password": "..."
4   }
```

Корисничко име и шифра су обавезни елементи захтева. Уколико је корисник унео исправне креденцијале, прослеђује му се следеће:

- Задаци које може да ради,
- Задаци које је претходно решавао,
- Питања која је поставио,

- Ранг листе,
- Обавештења.

Дохватање задатака за рад

Као што је поменуто у поглављу 4, дохватање задатака је оптимизовано тиме што се задаци чувају у документу, па се логовањем задаци само дохватају из њега.

Ова једноставна идеја је мало проширена да узме у обзир права приступа задацима. Ако би све задатке чували у једном документу онда би морали да филтрирамо задатке којима корисник има приступ пре него их пошаљемо.

Други приступ који је овде и имплементиран је да постоји више докумената са задацима. Сваки документ чува задатке за тачно одређену улогу. Тако да када корисник са улогом *2 - студент* захтева задатке њему се шаље садржај документа *2_file_for_user*.

Додатно треба обратити пажњу да ове документе треба освежавати по потреби. А посебно треба обратити пажњу на наредне случајеве:

- Када улога добије или изгуби приступ оцењивачу;
- Када се оцењивач активира или деактивира;
- Када се додају нови задаци или измене постојећи.

Дохватање претходно урађених задатака

Када се корисник улогује треба да има могућност да ради задатке, али треба и да му се прикажу задаци које је претходно радио. За разлику од претходно објашњених задатака за рад, овде нема смисла за сваког корисника у документу чувати његов претходни рад. Ти документи би захтевали компликовано одржавање и било би их колико и корисника. Према томе, то треба сваки пут дохватити из базе.

Овде ћемо део оптимизације пребацити на клијентску страну. Односно, када се корисник улогује добиће информацију о томе шта је претходно радио. Када корисник реши задатак, на клијентској страни ће бити измењена локална копија претходног рада студента. Замислимо да се корисник улоговао и да је добио низ од 5 успешно урађених задатака. Када корисник реши

шести задатак добиће одговор сервера да је задатак успешно решен. Пре него је сервер послао одговор, у бази је забележено да је корисник успешно решио тај задатак. Корисник ће када добије одговор у свој низ од 5 успешно решених задатака додати и шести. Нема потребе да сервер још једном приступа задацима да би кориснику јавио оно што он већ индиректно зна, односно да сада има 6 успешно решених задатака.

Додатна оптимизација која овде долази до изражаја када систем има више корисника је индексирање. Ово је већ образложено у делу 6.1.11.

Заједно са урађеним задацима ће бити дохваћена и питања која је корисник поставио за њих.

Одговор на логовање

Претходно је објашњено како се дохватају задаци за рад, претходно решавани задаци и питања. Обавештења и ранг листе се дохватају из одговарајућих докумената. Као што је претходно помињано, документи чувају податке из базе којима се често приступа.

Одговор на логовање је у наредном формату:

```
{
        "id": 1,
        "username": "...",
        "password": "...",
        "email": "...",
        "firstname": "...",
        "lastname": "..."
        "roleId": 4,
        "taskComponent": {
                "professors": [
                        {
                                "id": 2,
                                "username": "..."
                        }
                ],
                "tasks": [
                        {
```

```
                                "taskId": 1,
                                "graderId": 1,
                                "name": "...",
                                "text": "...",
                                "ordering": "..."
                        }
                ]
        },
        "submissions": [
                {
                        "taskId": 2,
                        "isWaitingForResponse": false,
                        "noTotalSubmissions": 3,
                        "noCorrect": 1,
                        "message": "...",
                        "isCorrect": true,
                        "question": {
                                "professor": "...",
                                "createdDate": "...",
                                "question": "...",
                                "response": "...",
                                "professorId": 2,
                                "userId": 1,
                        }
                }
        ],
        "stats": {
                "taskStats": [
                        {
                                "taskId": 1,
                                "task": "...",
                                "correctSubmissions": 3,
                                "totalSubmissions": 4,
                                "successPercantage": 100
```

```
52                          }
53                  ],
54                  "userStats": [
55                          {
56                                  "correctSubmissions": 4,
57                                  "userId": 1,
58                                  "user": "...",
59                                  "totalSubmissions": 8,
60                                  "successPercantage": 20
61                          }
62                  ]
63          },
64          "notifications": [
65                  {
66                          "id": 1,
67                          "title": "...",
68                          "text": "...",
69                          "createdDate": "..."
70                  }
71          ]
72  }
```

Како је одговор на логовање доста гломазан у наставку ће бити укратако образложени елементи:

- *id*, *username*, *password*, *email*, *firstname*, *lastname*, *roleid* - корисничке информације. Ове информације се приказују кориснику а он може да их мења. Клијентска апликација ће на основу идентификатора улоге одредити да ли треба да се прикаже страница за студенте или наставнике.
- *taskComponent* - представља објекат са информацијама потребним за задатке. Састоји се из листе наставника и листе задатака. Листа наставника је неопходна јер су то корисници којима може да се постави питање везано за задатке. Сваки задатак у листи задатака поред на-

зива, текста, идентификатора има и идентификатор оцењивача коме ће се проследити захтев за оцењивање.

- *submissions* - представља листу задатака које је корисник претходно радио. Овде се налази број покушаја, број успешних решења, порука која је добијена задњим послатим решењем, да ли је задње решење било тачно. Уколико је корисник поставио питање за рађени задатак унутар рада ће се наћи и *question* објекат. Овај објекат између осталог садржи текст питања, коме је постављено и одговор уколико постоји.

- *stats* - објекат који садржи две ранг листе, односно ранг листу корисника и ранг листу задатака. Ове вредности се прослеђују из погледа објашњеног у делу 6.1.9.

- *notifications* - низ најновијих обавештења која се приказују на страници. Свако обавештење садржи наслов, текст обавештења и време када је објављено.

6.2.3 Оцењивање задатака

Када је корисник улогован он може да ради задатке тако што ће их кликом на дугме послати на оцењивање. Тиме се серверу шаље POST захтев на */server/checkSolution* и садржи наредно тело захтева:

```
{
        "userId": 22,
        "taskId": 1,
        "graderId": 1,
        "solution": "",
        "ordering": "1"
}
```

Захтев садржи наредне елементе:

- *userId* - идентификатор корисника,

- *taskId* - идентификатор задатака,

- *graderId* - идентификатор оцењивача задуженог за оцењивање,
- *solution* - упит који треба оценити,
- *ordering* - сортирање уколико га треба додати. Одлучено је да се сортирање шаље као део захтева да се задатак не би дохватао из базе при сваком оцењивању само да би се видело сортирање. Клијентска апликација ту информацију неће приказати кориснику, али ће је искористити при слању захтева.

Када сервер прими захтев он га уноси у базу захтев и формира нови захтев формата објашњеног у делу 5.5.2, а затим га шаље одговарајућем оцењивачу. Када прими одговор оцењивача унеће га у базу, а затим проследити кориснику који је послао иницијални захтев.

6.2.4 Постављање питања

Серверу се шање POST захтев на */message/askQuestion* са наредним телом:

```
{
        "userId": 3,
        "professorId": 2,
        "taskId": 2,
        "question": ""
}
```

Тело захтева се састоји из наредних елемената:

- *userId* - идентификатор корисника који поставља питање,
- *professorId* - идентификатор наставника коме је постављено питање,
- *taskId* - идентификатор задатака за који се поставља питање,
- *question* - текст питања.

Овим захтевом ће се у базу унети питање и на то питање ће наставник касније моћи да одговори. Када корисник постави питање он неће моћи да ради тај задатак док не добије одговор или док не обрише питање.

6.2.5 Брисање питања

Као што је претходно речено, корисник не може да ради задатак уколико је поставио питање за њега, а још увек нема одговор. Уколико се корисник предомисли може да обрише питање и настави да ради задатак. Захтев за брисање питања се шаље POST методом на */message/deleteQuestion* са наредним телом:

```
{
        "userId": 3,
        "taskId": 2
}
```

Тело захтева се састоји из наредних елемената:

- *userId* - идентификатор корисника,
- *taskId* - идентификатор задатака.

Уколико је наставник већ одговорио није могуће обрисати питање, али ће корисник моћи да настави да решава задатак.

6.2.6 Додавање задатака

Додавање нових задатака се врши слањем POST захтева са одговарајућим телом на */server/addTasks*:

```
{
        "graderId": 1,
        "tasks": [
                {
                        "name": "...",
                        "task": "...",
                        "solution": "...",
                        "ordering": "..."
                }
```

```
10          ]
11  }
```

- *graderId* - представља јединствени идентификатор оцењивача коме су задаци намењени;
- *tasks* - низ са задацима које треба унети;
- *name* - назив задатка;
- *task* - текст задатка;
- *solution* - тачно решење задатка;
- *ordering* - сортирање које треба додати на крај.

в захтев прво ће унети задатке у табелу задатака. Затим ће послати захтев одговарајућем оцењивачу да изврши генерисање резултата на основу прослеђеног тачног решења. Када оцењивач одговори серверу он ће одговорити на првобитни захтев корисника. Одговор сервера ће укључивати све грешке које су се у међувремену десиле, као и целокупан одговор оцењивача. Одговор ће бити у наредном формату:

```
1   {
2   "graderResponse": {
3           "totalTime": "...",
4           "tasks": [
5                   {
6                           "noRows": "...",
7                           "time": "...",
8                           "taskId": "..."
9                   },
10          ]
11  },
12  "errors": []
13  }
```

Одговор се састоји од следећих елемената:

- *graderResponse* представља целокупан одговор оцењивача као што је описано у делу за генерисање решења 5.5.1;
- *errors* представља низ грешака до којих је дошло на серверу, на пример уколико је у захтеву био задатак без тачног решења, или уколико је захтев послат неактивном оцењивачу, или оцењивач не постоји.

6.2.7 Измене задатака

Очекивано је да ће у неким случајевима бити потребна измена задатка, на пример измена текста или самог решења. Када се то деси биће потребно поново генерисати решење у оцењивачу. Као што је претходно наведено, оцењивач не садржи текст задатка ни његово решење, већ само документ тачно одређеног назива који садржи резултат извршавања тачног упита. Још један разлог због ког би регенерација задатака била потребна је у случају да су ови документи у оцењивачу из неког разлога изгубљени.

Серверу се може послати захтев за измену задатака на адресу */server/updateTasks*. Тело захтева је идентично захтеву за унос нових задатака, са разликом да постоји аргумент *taskId*, и да су све вредности осим идентификатора опционе.

```
{
        "graderId": 1,
        "tasks": [
                {
                        "taskId": 1,
                        "name": "...",
                        "task": "...",
                        "solution": "...",
                        "ordering": "..."
                }
        ]
}
```

Све вредности које су прослеђене ће бити и измењене у табели. Уколико на пример *name* није наведен неће се мењати. Вредност *taskId* је једини обавезан аргумент и представља јединствени идентификатор задатка. Уколико се задатку проследи само *taskId* задатак неће бити промењен већ ће на основу података у бази бити формиран захтев за генерисање резултата и бити послат оцењивачу.

Одговор сервера у овом случају ће бити идентичан одговору који се добије слањем захтева за унос задатака. Односно садржаће целокупан одговор оцењивача и списак евентуалних грешака које су се десиле на серверу.

6.2.8 Одговор на питања

Наставник има могућност да одговори на студентско питање слањем POST захтева на */message/respondToQuestion*, захтев треба да има наредно тело:

```
{
        "userId": 1,
        "professorId": 2,
        "taskId": 3,
        "response": ""
}
```

Тело захтева се састоји од

- *userId* - идентификатор корисника који је поставио питање,
- *professorId* - идентификатор наставника коме је постављено питање,
- *taskId* - идентификатор задатака,
- *response* - текст одговора на питање.

6.2.9 Постављање обавештења

Наставник има могућност да постави обавештење слањем POST захтева на */notification/new*, захтев треба да има наредно тело:

```
{
        "title": "",
        "text": ""
}
```

Тело захтева се састоји од

- *title* - наслов обавештења,
- *text* - текст обавештења.

Када сервер прими овакав захтев, у базу ће бити унето ново обавештење. Након тога ће бити освежен документ са обавештењима. Као што је поменуто у поглављу 4, помоћни документ ће садржати одређени број најновијих обавештења да се бази не би приступало при сваком логовању корисника. Обавештења су за све кориснике иста. Обавештења су углавном информативног типа, попут унапређења система и нових функционалности. По потреби администратор може да обрише застарела обавештења.

Глава 7

Имплементација клијента

Као што је наведено у делу 4.2 клијентска апликација је имплементирана у *Angular*-у. Да би корисник могао да користи овај систем потребно је прво да се улогује. У зависности од тога који је корисник улогован разликујемо два профила, студент и наставник. Студенти имају могућност да раде задатке и постављају питања, док наставници имају могућност да додају задатке и одговарају на питања. Наставници такође имају могућност као и студенти да раде задатке и да постављају питања.

У делу 6.1.1 је наведено да оцењивач разликује 4 улоге: *администратор*, *професор*, *студент* и *остали*. Профил студент ће имати корисници са улогом *студент* или *остали*. Ове две улоге имају идентичне намене, док се разликују само у томе којим задацима имају приступ. Улоге *администратор* и *професор* ће имати додељен профил *наставник*.

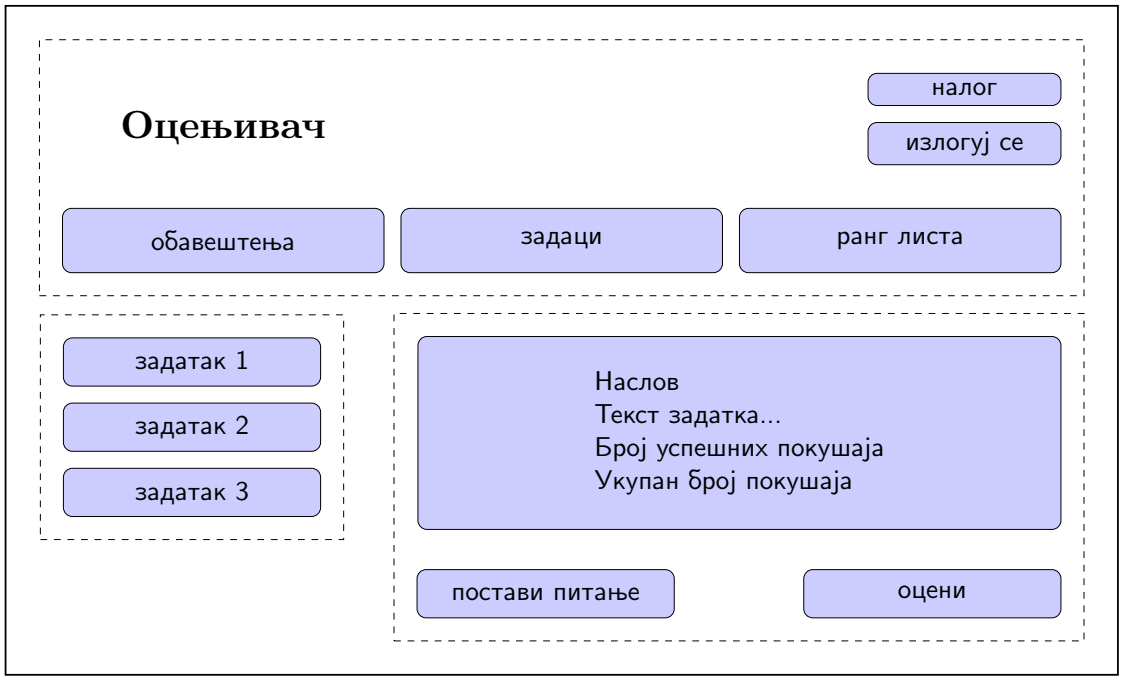
Одлучено је да корисници са улогом *администратор* немају засебан профил за приказ јер ће они имати директан приступ систему. Односно, администратор ће директно приступити бази података и извршити измене. Како администратор има задатак да одржава систем нема претерано смисла правити приказ за све оно што би се једноставније урадило директним приступом. Администратор ће имати исти профил као и наставник из разлога што су његова задужења ближа наставничким него студентским.

7.1 Студентски профил

Апликација има могућност да прикаже наредне елементе за студентски профил:

- Обавештења,
- Списак доступних задатака,
- Задатак изабран за рад, односно његов текст и резултате рада,
- Кориснички налог, са могућношћу да се подаци измене,
- Ранг листу.

Распоред елемената главног приказа је дат на слици 7.1. Страница је додатно стилизована у имплементацији.



Слика 7.1: Распоред на страници са задацима

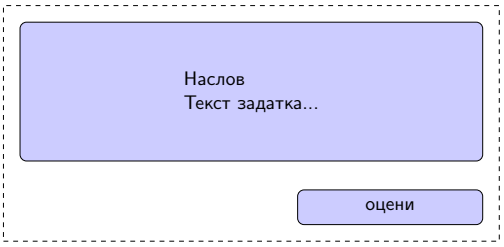
Улогован корисник има опције да се излогује (дугме *излогуј се*) и да прикаже информације о свом налогу (дугме *налог*).

Обавештења, *задаци* и *ранг листа* чине мени. Кликом на *обавештења* ће бити приказан списак обавештења, а ранг листа кликом на дугме *ранг листа*. Приказ на слици 7.1 се добија кликом на *задаци* и представља страницу са задацима.

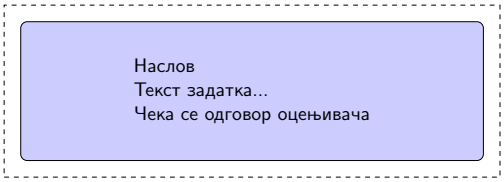
Страница са задацима са леве стране има списак свих задатака који су доступни кориснику за рад. Када корисник кликне на жељени задатак добиће додатне информације о задатку. Задатак који се приказује има наслов и

текст. У зависности од тога ког је статуса задатак за улогованог корисника биће различит приказ. Разликујемо наредне случајеве:

- Корисник претходно није радио задатак. У овом случају ће бити приказани само наслов, текст и могућност да корисник пошаље задатак на оцењивање. Приказано на слици 7.2.
- Корисник је послао задатак на оцењивање али још увек није добио одговор оцењивача. У овом случају је приказ идентичан претходном, са разликом да корисник више нема могућност да пошаље задатак на оцењивање. Овим се ограничава да корисник А може да има задатак Б на оцењивању само једном у сваком тренутку. Приказано на слици 7.3.
- Корисник је добио одговор оцењивача за свој рад. У овом случају је приказ идентичан почетном, са разликом да сада има могућност да постави питање наставнику. Поред тога, уколико је корисник радио задатак бар једном биће му приказано колико пута је успешно решио задатак, колико укупно пута је покушао да реши задатак као и који је одговор добио при последњем решавању. Приказано на слици 7.4.
- Корисник је поставио питање наставнику. У овом случају корисник не може да ради задатак док или не добије одговор наставника или не обрише питање. Ово је приказано на слици 7.5.

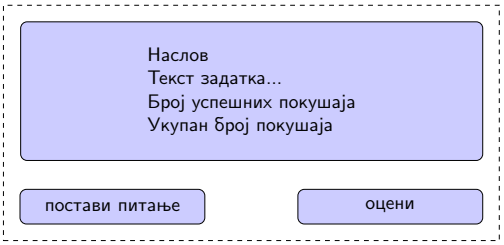


Слика 7.2: Задатак није рађен

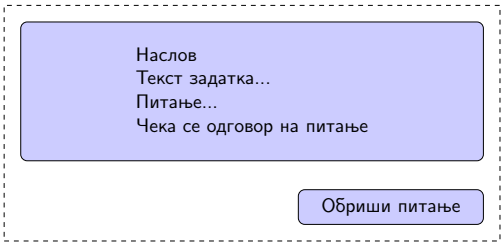


Слика 7.3: Задатак послат на оцењивање

Приказ осталих страница, односно почетна страница за логовање, обавештења, ранг листе и налог, неће бити додатно разматрани јер само приказују одговарајуће информације и не садрже комплекснију логику.



Слика 7.4: Корисник је радио задатке



Слика 7.5: Постављено је питање

7.2 Наставнички профил

Апликација има могућност да прикаже наредне елементе за наставнички профил:

- Обавештења са формуларом за додавањем новог обавештења,
- Списак доступних задатака, са могућношћу измене постојећег и додавање новог задатка,
- Ранг листу,
- Списак свих питања постављених наставнику, са могућношћу да се одговори,
- Кориснички налог, са могућношћу да се подаци измене.

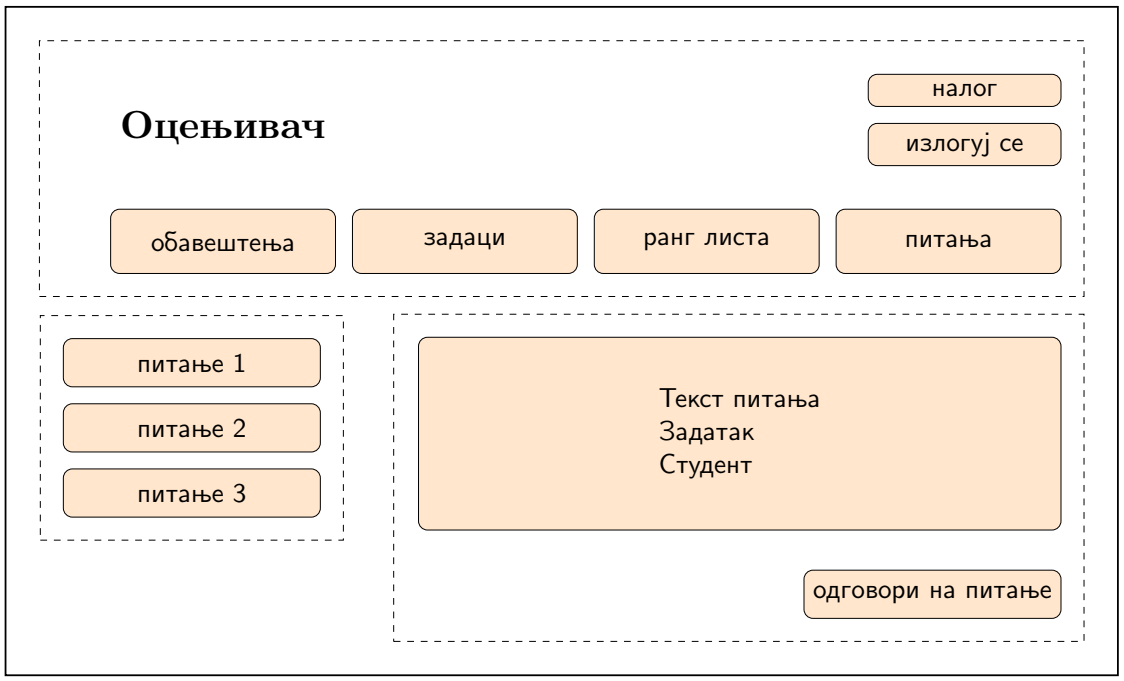
На слици 7.6 је приказан распоред који наставник види у делу за питања.

Наставник има могућност да ради задатке као и студенти. Како је то идентично студентском налогу овде неће бити поново разматрано.

7.3 Оптимизација на клијентској страни

Као што је описано у делу 6.2.2, део оптимизације је са серверске стране пребачено на клијентску страну. Клијент не треба да шаље серверу непотребне захтеве. Наиме, када се корисник први пут улогује потребно да добије наредне информације:

- Информације о кориснику. Логовање се врши са корисничким именом и шифром, након логовања корисник треба да види своје име, презиме и мејл.



Слика 7.6: Распоред на страници са питањима

- Обавештења.
- Задатке које корисник може да ради.
- Информације о задацима које је корисник претходно радио.
- Ранг листу.

Једном када се корисник улогује остаје улогован одређено време. Након тога је потребно да се опет улогује и дохвати све податке поново.

Можемо претпоставити да се доступни задаци и обавештења неће често мењати, и да је у реду да корисник добије информације о променама које су наступиле након последњег логовања при наредном логовању. Односно, уколико је постављено ново обавештење прихватљиво је да га корисник добије након што се наредни пут улогује.

Ранг листа се мења сваким успешним решавањем задатака. Дохватање ранг листе након сваког успешног решавања задатка је неефикасно. Треба имати у виду да паралелно више корисника ради задатке, и да када један корисник успешно реши задатак сачувана ранг листа другог корисника нема тачне информације. Слично претходном, корисник ће нову ранг листу добити тек када се следећи пут улогује.

Информације о урађеним задацима ће бити одржаване и од стране клијента. Односно, уколико клијент добије одговор од сервера да је успешно решио задатак повећаће број успешно решених задатака за 1. Нема потребе да поново контактира сервер да би добио информацију о броју успешних решавања.

Глава 8

Докеризација

8.1 Докеризација оцењивача

Као што је наведено у почетку, цео оцењивач треба да ради у једном докер контејнеру. При докеризацији је потребно обратити пажњу на неколико ствари о којима ће бити речи у наставку.

У конкретној имплементацији са системом за управљање базом података *Db2* и *Java* апликацијом је постојало неколико начина за докеризацију:

1. Кренути од произвољне *Linux* слике и инсталирати *Db2* и *Javu*;
2. Кренути од доступне *Java* слике и у њој инсталирати *Db2*;
3. Кренути од доступне *Db2* слике и у њој инсталирати *Javu*.

Прва могућност је одмах одбачена јер је једноставније кренути са бар једним инсталираним програмом него инсталирати оба. Друга опција је одбачена јер се подешавање система *Db2* показало знатно комплексније од инсталирања језика *Java*. Према томе, за имплементацију је изабрана трећа могућност.

Поред претходно наведеног у контејнеру треба да се нађу још неки документи и директоријуми да би оцењивач функционисао. Ради једноставније структуре унутар контејнера одлучено је да се све ово нађе у *home* директоријуму. Дакле, *home* директоријум треба да садржи наредно:

- *grader.jar* - извршиви фајл који је централни део сервиса оцењивача - *контролер оцењивача*, име фајла може да варира,

- Директоријум *scripts* са скриптама *create_ results.sh* и *check_ solution.sh*. Прва ће бити коришћена за генерисање докумената са тачним резултатом, а друга за оцењивање задатака. Њихов садржај је објашњен у поглављу 5.4.

- Директоријум *results* у ком ће се наћи тачни резултати,

- Директоријум *user_ results* у ком ће се привремено налазити кориснички резултати, односно док се не обришу на крају оцењивања,

- Директоријум *stud2020* који треба да садржи све неопходно за креирање базе, унос података и давање одговарајућих права приступа.

Важна компонента у докер контејнеру је база података, она треба да буде подешена са ограничењима приступа наведеним у поглављу 5.2. У истом поглављу је наговештено да је у докер контејнеру потребно направити новог корисника који ће имати ограничена права приступа.

Постојала је идеја да се направи захтев чијим би слањем била креирана или регенерисана база у докер контејнеру. То је имплементирано, али се показало као непрактично. Наиме, креирање базе података може изазвати разне грешке, те би најбоље било то урадити одвојено без аутоматизације. С обзиром на то да се база креира једном на почетку, да се њена структура неће мењати и да је база заштићена, те да је корисници неће пореметити ово је прихватљив приступ.

Одвојено креирање базе података подразумева у докер контејнеру извршити одговарајуће команде. У конкретном случају би то било:

```
1        docker exec -it grader bin/bash
2        su - db2inst1
3        cd /home/stud2020
4        ./create.sh
```

Претходна команда се састоји из следећих елемената:

- *docker exec* је команда којом се улази у одговарајући докер контејнер,

- *su*, скраћено од *switch user* је команда којом се мења корисник; овде је потребно да то буде корисник са привилегијама администратора, односно *db2inst1*,
- *cd* је команда за позиционирање у одговарајући директоријум,
- *./create.sh* је команда за покретање скрипте за креирање базе података; ово извршавање може да потраје али је важно испратити због евентуалних грешака.

Докер контејнер у ком се налази оцењивач ће имати три корисника:

- *root* - подразумевани корисник са свим привилегијама, али без приступа *db2* систему,
- *db2inst1* - корисник који има приступ *db2* систему за управљање базом података,
- *student* - корисник са ограниченим правима приступа бази, креирање корисника је образложено у делу 5.2.

Извршавање *grader.jar* фајла је доступно свим корисницима, али база није. Како апликација треба да има приступ бази њу мора да покрене корисник који има права приступа истој. Дакле, апликацију за оцењивање треба да покреће *db2inst1* корисник.

Овде настаје проблем када је *root* власник директоријума које апликација користи. Апликацију извршава *db2inst1* корисник, па уколико директоријуми немају одговарајућа права приступа, апликација неће моћи да направи документе са тачним резултатима, самим тим ни да оцени корисничке радове. Ово је решено променом власника директоријума и прављењем поменутих директоријума од стране одговарајућег корисника. Односно, решава се покретањем апликације на следећи начин:

```
1         chown db2inst1 home
2         sudo su - db2inst1 -c "mkdir -p /home/user_results"
3         sudo su - db2inst1 -c "mkdir -p /home/results"
4         sudo su - db2inst1 -c "cd /home && java -jar grader.jar"
```

8.2 Докеризација сервера

Контејнер у ком ће радити сервер описан у поглављу 6 треба да садржи *Java* апликацију која представља контролер, базу података и помоћне документе.

Као и у случају оцењивача, и овде се креће од *Db2 docker* слике и у њу се инсталира *Java* окружење. За разлику од оцењивача, овде није потребан додатни корисник, већ су *root* и *db2inst1* довољни.

Приликом конфигурације контејнера је потребно покренути скрипту која ће креирати базу података, табеле, погледе, окидач и индексе објашњене у поглављу 6.1. Приликом креирања базе се уносе подразумеване улоге корисника, док се остали подаци, попут оцењивача, додају по потреби.

Помоћне документе креира и одржава контролер. Нако што се апликација контролера покрене она ће иницијализовати све потребне документе. Помоћни документи подразумевају документ са доступним задацима, обавештења и ранг листе.

8.3 Докеризација клијента

Радни оквир *Angular* има своју званичну докер слику на основу које се креира контејнер. Докеризација подразумева копирање неопходних докумената у контејнер приликом његовог креирања.

Глава 9

Предлози за унапређења система

У овом делу ће бити представљено неколико начина да се овај систем унапреди. Прво ће бити размотрена параметризација упита, чиме би се повећала поузданост оцењивача. Након тога ће бити приказан један приступ проширења скупа упита погодних за оцењивање и на оне у којима може да се мења структура база и садржај табела. На крају ће бити продискутовано унапређење у погледу броја паралелних захтева који могу да се извршавају.

9.1 Параметризација упита

Оцењивач представљен у овом раду не проверава да ли је корисник радио задатак на исправан начин. Нека је дат задатак у ком се тражи да корисник издвоји све испите на којима је полаган предмет *Relacione baze podataka*. Исправан начин да се ово реши би био да се табела *ISPIT* споји са табелом *PREDMET*, одакле може да се провери назив предмета, односно:

```
SELECT I.*
FROM DA.ISPIT I JOIN DA.PREDMET P
        ON P.ID=I.IDPREDMETA
WHERE P.NAZIV='Relacione baze podataka'
```

Уколико корисник зна да је идентификатор предмета 2016 исти резултат

може да добије наредним упитом:

```
SELECT *
FROM DA.ISPIT
WHERE IDPREDMETA=2016
```

Иако оба упита дају исти резултат, први је отпоран на промене идентификатора у бази, што га чини исправнијим. Корисници не могу да виде који су идентификатори предмета у бази, односно не могу да виде податке у бази, па се тиме повећава шанса да ће корисници писати упите на исправан начин.

Уколико би било потребно додатно проверити да ли је корисник написао логички исправан упит могла би се додати параметризација. На пример, у задатку би уместо „из предмет *Relacione baze podataka*" могло да се тражи „из предмета са називом *SECRET_PREDMET*". У процесу оцењивања би се у упитима *SECRET_PREDMET* заменило са једним или више конкретних предмета. Упит би се извршио по једном за сваки од различитим параметара, и проверавао би се резултат добијен сваким од извршавања.

Ово је релативно једноставно проширење система, а може знатно повећати поузданост у исправност корисничког решења.

9.2 Проширење скупа *SQL* упита које је могуће прегледати

Оцењивач представљен у овом раду је ограничен на оцењивање *SQL* упита којим се дохватају информације из базе. Овим оцењивачем није могуће тестирати креирање табела, погледа, функција окидача и слично. Штавише, база над којом се ради је обезбеђена да корисници не могу да креирају ове структуре. Систем је могуће проширити да се ово омогући, али то са собом носи извесне проблеме.

На пример, претпоставимо да имамо једну базу и задатак да се креира табела са одређеним колонама. Уколико имамо два корисника који истовремено раде овај задатак, један од њих ће добити грешку јер табела већ постоји и креирана је од стране другог корисника. Слични проблеми се јављају и при креирању окидача и функција. Чак и ако корисници овакав задатак не раде

истовремено, треба водити рачуна да се база врати у првобитно стање да би наредни корисник могао да ради исти задатак.

Најједноставнији начин да се претходно реши је да у једном тренутку само један корисник може да ради задатак који подразумева модификацију базе, и да се након његовог рада база враћа у првобитно стање. Један начин за освежавање базе би био брисање базе и њено поновно креирање, али то је доста неефикасно. Други начин би подразумевао вођење евиденција о изменама и њихово поништавање.

Претходни приступ је доста неефикасан, али би се могао оптимизовати уколико не би имали једну овакву базу, већ више њих. Тиме би корисник добио на коришћење прву слободну базу, док би чекао уколико нема слободних. Додатно би се могла размотрити могућност коришћења различитих схема. У овом приступу би база садржала, на пример, 10 идентичних схема. На тај начин би у једном тренутку задатке могло да ради 10 корисника, по један у свакој схеми. Након оцењивања задатка би се освежавале само табеле једне схеме, што је ефикасније од освежавања целе базе.

Уколико би био решен проблем извршавањем упита који модификују базу, поставило би се питање о провери тачности решења. Наиме, уколико се у задатку тражи креирање табеле, како би се проверило да ли је табела исправно креирана. Након извршавања корисничког упита за који се очекује да креира табелу може се извршити скрипта која би покушала да унесе редове у нову табелу. Уколико унос редова не успе то би значило да табела није исправно креирана. Или, на пример, уколико се у задатку тражи креирање окидача, скрипта за проверу задатка би мењала податке за које се очекује да окину окидач, а затим би проверавала да ли је окидач стварно урадио оно што се тражи у задатку.

Идеја је да се у оваквим случајевима изврши скрипта за проверу тачности креираних структура. Оно што та скрипта може да уради је да изврши команде над базом. Излаз скрипте је оно што треба испратити и шта индицира да ли је дошло до грешке. Дакле, постојао би очекиван излаз за извршавање скрипте за проверу, а та скрипта би се извршавала након извршавања корисничког упита који треба да креира тражене структуре. Тај излаз би се проверавао са *diff*, идентично проверавању тачности резултата извршавања упита претходно описаних у овом раду.

9.3 Динамичко одређивање броја захтева

Сервер и оцењивач представљени у овом раду могу да обрађују одређени број захтева у једном тренутку, и тај број не може да се мења у току извршавања. За сервер то није експлицитно ограничено, већ библиотека то одређује у складу са могућностима рачунара на ком се покреће. Са друге стране, оцењивачу се експлицитно поставља број нити. Побољшање би могло да се огледа у томе да се број захтева који могу да се обрађују у једном тренутку динамички одређује и мења у зависности од оптерећења система.

Глава 10

Закључак

У овом раду је представљен систем намењен оцењивању задатака из *SQL-a*. Систем представља једну микросервисну апликацију која се састоји из клијента, сервера и оцењивача. Систем је скалабилан и има могућност да опслужује већи број корисника истовремено.

Оцењивач представљен у овом раду је показатељ да је могуће аутоматско оцењивање задатка из *SQL*-а, са ограничењем да су то задаци у којима се тражи издвајање података. Оцењивач представљен овде је доста строг и задаци морају испуњавати захтеве наведене у поглављу 5.7.

Коришћење микросервисне архитектуре у овом раду показало се као добар избор. Наиме, прво је у потпуности имплементиран оцењивач, а затим је тестиран као једна целина. Након имплементације сервера је успостављена комуникација са оцењивачем, и систем састављен од та два сервиса је тестиран. Због оваквог начина рада, током имплементације клијента се могло претпоставити да остатак система ради исправно. Додатно, тестирани сервиси су контејнеризовани коришћењем алата *Docker*, чиме је олакшано њихово покретање. На пример, оцењивач је био покренут у контејнеру током имплементације клијентске апликације.

Објектно релационо мапирање које нуди *Hibernate* се показало изузетно корисно током имплементације. Наиме, непосредно пред крај имплементације је било потребно променити називе табела и колона како би формат био конзистентан. Захваљујући овој библиотеци, није било потребно променити називе у великом броју упита, већ су промењене само вредности у класама које представљају табеле.

Количина ресурса које СУБП *Db2* користи при извршавању упита, попут

времена извршавања и процесорских ресурса, није иста при сваком извршавању упита. Наиме, није могуће гарантовати да ће извршавање једног упита увек заузети исту количину ресурса, шта више, мала одступања су очекивана. Како је било неопходно ограничити ресурсе додељене за извршавање упита, конзистентност коришћења ресурса од стране СУБП је била кључна. Паметним избором ограничења може се очекивати да ће сви ефикасни упити бити извршени без прекорачења ограничења, док ће неефикасни упити бити прекинути.

Систем је имплементиран на начин да је релативно лако додати још оцењивача. За додавање оцењивача за други СУБП се може користити имплементација из дела 5, или се правити на другачији начин.

Могуће је додати оцењиваче и за друге програмске језике, као на пример језике *Python*, *Java* и *C*. У овом случају ће оцењивач изгледати знатно другачије, јер извршавањем оваквих кодова треба заштитити систем на другачији начин. Додатни оцењивачи не морају бити имплементирани у језику *Java*, већ се може користити произвољан програмски језик.

Оцењивач представљен овде није намењен за оцењивање задатака на испиту. Као што је претходно поменуто, оцењивач је строг и даје одговор само да ли је решење тачно или не, док је оцењивање испитних радова другачије. Наиме, на испиту студенти добијају одговарајући број бодова у складу са деловима упита који су тачни. Студенти могу користити оцењивач током припреме за испит како би проверили своје знање.

Библиографија

[1] Angular, 2024. on-line at: https://angular.io/.

[2] Db2, 2024. on-line at: https://www.ibm.com/products/db2.

[3] DB2 Docker, 2024. on-line at: https://www.ibm.com/docs/en/db2/11.5?topic=deployments-db2-community-edition-docker.

[4] Docker, 2024. on-line at: https://www.docker.com/.

[5] Docker Swarm, 2024. on-line at: https://docs.docker.com/engine/swarm/.

[6] Hibernate, 2024. on-line at: https://hibernate.org/.

[7] JDBC, 2024. on-line at: https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/.

[8] Kubernetes, 2024. on-line at: https://kubernetes.io/.

[9] Maven, 2024. on-line at: https://maven.apache.org/.

[10] MySQL vs IBM DB2: 6 Critical Differences, 2024. on-line at: https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/.

[11] Petlja, 2024. on-line at: https://petlja.org/sr-Latn-RS/.

[12] TIOBE, 2024. on-line at: https://www.tiobe.com/tiobe-index/java/.

[13] TypeScript, 2024. on-line at: https://www.typescriptlang.org/.

[14] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, jun 1970.

[15] Eclipse Foundation. Grizzly Best Practices, 2016. on-line at: `https://javaee.github.io/grizzly/coreconfig.html`.

[16] Eclipse Foundation. Grizzly dokumentacija, 2024. on-line at: `https://javaee.github.io/grizzly/index.html`.

[17] HGInsights. HGInsights for IBM Db2, 2024. on-line at: `https://discovery.hgdata.com/product/ibm-db2`.

[18] Oracle. Java, 2024. on-line at: `https://www.java.com/en/`.

[19] Огњен Коцић. Оцењивач задатака из програмирања - имплементација сложеног софтвера у модерном C++-у. 2015.

Биографија аутора

Милица Гњатовић је рођена 18. маја 1999. године у Београду. Основне студије је завршила на Математичком факултету у Београду, на смеру информатика 2022. године са просечном оценом 9.30. Тренутно је студент мастер студија на истом факултету.

Милица је радила као демонстратор на Математичком факултету у летњем семестру школске 2021/2022. Од зимског семестра школске 2022/2023 је сарадник у настави на Математичком факултету, на катедри за рачунарство и информатику. Она предаје предмете *Увод и веб и интернет технологије* и *Релационе базе података*.

Од 2022. године Милица ради као *Salesforce Developer* у компанији *Deloitte*.