Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Рачунарска техника и софтверско инжењерство

Ниво студија: Основне академске студије

Предмет: Пројектовање информационог система и базе података

Број индекса: 556/2016

Лазар Д. Марковић

Базе података и системи за управљање процесима

дипломски рад

Комисија за преглед и одбрану:

1. др Милан Д. Ерић – ментор Датум одбране:_____

2._____ Оцена:_____

3._____

Универзитет у Крагујевцу
Факултет инжењерских наука

Основне студије: **Рачунарска техника и софтверско инжењерство**

Име и презиме: **Лазар Марковић**

Број индекса: **556/2016**

ПРИЈАВА ДИПЛОМСКОГ РАДА

Тема рада: ***Базе података и системи за управљање процесима***

Задатак: У оквиру дипломског рада обухватити: Уводна разматрања у вези пројектовања базе података; Рад са базом података; Опис процеса управљања; Конкретан пример примене; Закључна разматрања.

Ментор:

Проф. др Милан Д. Ерић

Крагујевац, 13.03.2020.

Изјава о самосталној изради рада

Изјављујем да сам овај дипломски рад урадио самостално, служећи се стеченим знањем и искуством као и наведеном литературом.

Лазар Марковић 556/2016

_____

(потпис)

Садржај

The summary

This documentation presents a database and a process management system. A description of the database, the basics of the database system, work with basic data and an overview of the subject that contains the database are presented. A description of the process of managing and inflating projects, designing applications for warehouse work is given.

Keywords: database, sql

Резиме

Овим радом приказана ја база података и систем за управљање процесима. Представљен је опис базе података, основе система базе података, рад са базом података и преглед објекта које база садржи. Дат је опис процеса управљања и упутство коришћења пројектоване апликације за магацинско пословање.

Кључне речи: база података, *sql*

1.Увод

Појам базе података појавио се крајем шездесетих година 20. века и означавао је скуп међусобно повезаних података који се чувају заједно, и међу којима има само онолико понављања колико је неопходно за њихово оптимално коришћење при вишекорисничком раду. Подаци се памте тако да буду независни од програма који их користе, и структуирају се тако да је омогућен пораст базе. После сваке активности над базом, која представља логичку целину посла, стање базе мора бити конзистентно. Дакле, подаци у бази и односи међу њима морају задовољити унапред задате услове који осликавају део реалности моделиране подацима у бази.

У развоју система база података може се уочити неколико генерација система за управљање базама података *(SUBP),* које су или коегзистирале на тржишту или смењивале једна другу. Фундаментална разлика између система ових генерација је разлика у моделу података, која се у имплементацији одговарајућих *SUBP* рефлектује на ефикасност приступа подацима и обраде података, продуктивност корисника, функционалност система и подршку разноврсним апликацијама. Тако се у прве две генерације сврставају такозвани мрежни *(CODASYL)* системи и хијерархијски системи, који су готово у потпуности превазиђени, осамдесетих година прошлог века, релационом технологијом као трећом генерацијом *SUBP*. Све три генерације намењене су пре свега пословно-оријентисаним апликацијама.

Архитектура највећег броја система база података одговара предлогу *ANSI/SPARC* студијске групе Америчког националног института за стандарде, и позната је као *ANSI* архитектура. Ова архитектура представљена је хијерархијом апстракција, при чему сваки ниво хијерархије укључује специфични начин представљања, репрезентацију објеката, односа међу објектима и операција над објектима. Хијерархијска архитектура омогућује природну декомпозицију и ефикасни развој система за управљање базама података.

У наставку рада биће описана разматрања у вези пројектовања базе података, рад са базом података, опис процеса управљања и конкретан пример примене пројектоване базе.

2. Рад са базом података

2.1 Опис базе података

База података представља колекцију међусобно повезаних података који су организовани у табеле и друге структуре података, а користе једну или више апликација. Основна намена базе података је да буде репозиторијум (складиште) за податке. Подаци могу бити различитог типа, текстуални, нумерички, слике, аудио и видео записи и сл. Подаци у базама података су организовани у дводимензионалне табеле. Табела може да има више колона, где свака колона представља неку особину или атрибут. Врсте табеле чине конкретни подаци, односно конкренте вредности особина/атрибута неког објекта. На пример, једна табела може да садржи информације о ученицима. Колоне табеле могу да дефинишу име, презиме, годину рођења ученика, и сл. Врсте у таквој табели су ученици, тако да се свака врста односи на једног ученика. Које ће табеле да садржи база података зависи од проблема за који треба реализовати базу података.

На пример, база података се може одности на школу, па ће у том случају табеле бити о ученицима, наставницима, одељењима, и сл. Поступак избора и дефинисања табела за базу података је део процеса моделирања односно изградње модела података. Међусобна повезаност података је оно по чему се база података разликује у односу на фајл системе (датотеке) и програме за унакрсна израчунавања као што је *Excel*. Повезаност података обезбеђује значајне предности код претраживања када корисник може да на основу веза извуче много више података. На пример, ако постоји табела која чува податке о ученицима и табела са подацима о одељењима, веза између ученика и одељења може да обезбеди да одговарајућим захтевом (*SQL* упитом) прикажете све ученике жељеног одељења. База података садржи и тзв. метаподатке, односно податке о самој структури базе података. Метаподаци могу да се односе на имена табела, имена колона у свакој табели, на податке о корисницима података, као и разним помоћним структурама које обезбеђују брз прступ подацима (индекси).

2.2 *SQL* језик

За рад с релацијском базом података користе се упитни језици. Данашњи упитни језици који се користе су најчешће процедурални и непроцедурални. За релацијску базу података се користе непроцедурални језици. Постоји више непроцедуралних језика, а највише се користи *SQL* језик. Иако се назива упитним језиком, *SQL* има наредбе за свеобухватан рад с релацијском базом података, а то су:

- креирање таблице
- унос података у таблице
- брисање података из таблица
- наредбе за дефинисање базе података (*DDL-Data Definition Language*)
- наредбе за манипулацију подацима (*DML-Data Manipulation Language*)
- управљачке наредбе.

SQL (***Structured Query Language***) , тј, његов развој је започео 1974. године, када је објављен чланак *D.D.Chamberlin* и *R.F.Boyce* у којем они описују упитни језик за претраживање под називом *SEQUL*. Након тога 1975. им се придружује и *M.Hammer*, те они заједничко представљају концепт језика *SQUARE*. Убрзо након тога *SQUARE* мења назив у *SEQUEL2*, и тај језик је коришћен у развоју првог прототипа релацијског састава за управљање базама података који је имао назив *«System P»*, а касније тај језик мења име у *SQL*. *SQL* је релацијски потпун јер за свих 5 основних релацијских оператора постављао је семантички еквивалентне *SQL* наредбе. *SQL* језик је стандардизован од *ISO* (1986.г.) и *ANZI* (1986.године) института и данас се ради у већини релацијских система базе података. *SQL* омогућава прављење упита над више релација истовремено. Релацијска база података спада у савремене базе података уз објектни или димензијски модел. *SQL* је декларативан језик у којем корисник специфира само жељени резултат. *SQL* 1992 подупире само функције које крајњи корисник жели, али такође подупире и оне функције које обрађују апстрактне типове података.

Апстрактни типови података се данас могу додавати или брисати из система, без да то има икаквог утицаја на сам састав. Иако такав приступ повећава целокупну функционалност састава, сами састави нуде веома ограничену потпору за оптимисање операција *SQL*-стандардни упитни језик.

2.3 Основе система *RDBMS*

Савремени напредни системи *RDBMS* не само да складиште податке, већ и управљају њима тако што ограничавају податке који могу ући у систем, а такође омогућују узимање података из система. Ако вам је потребно само да сместите податке негде на сигурно, можете да користите било какав систем за складиштење података. Системи *RDBMS* вам омогућују да одете изнад самог складиштења података у домен дефинисања начина на који би ти подаци требало да изгледају или пословних правила за податке. Не треба мешати оно што се назива „пословним правилима за податке" са генерализованим пословним правилима која управљају целокупним системом (на пример, неко не може ништа да види док се не пријави на систем или аутоматско прилагођавање текућег периода у рачуноводственом систему првог у месецу). Ти типови правила у ствари се могу применити на било ком нивоу система (у данашње време то се обично примењује у средњем или клијентском слоју вишеслојног система). Уместо тога, оно по чему се овде говори су пословна правила која се посебно односе на податке. На пример, не може постојати поруџбина са негативном количином. Код система *RDBMS* та правила могу да се угради директно у интегритет саме базе података.

2.4 Преглед објеката базе података

Систем *RDBMS* као што је *SQL* сервер садржи велики број објеката. Код *SQL* сервера списак неких важнијих објеката базе података садржи ствари као што су:

- сама база података
- дневник трансакција
- склопови
- табеле
- групе датотека
- ускладиштене процедуре
- приказ
- извештаји
- функције које дефинише корисник
- корисници
- роле

2.4.1 Базе података као објекат

База података је у ствари објекат највишег нивоа који може да се референцира у оквиру датог *SQL* сервера. (Технички говорећи, и сам сервер се може сматрати објектом, али не из било какве реалне „програмерске" перспективе.) Већина других објеката *SQL* сервера, али не сви, изведени су од објекта базе података.

База података обично представља групу која садржи барем скуп објеката табела и најчешће друге објекте као што су ускладиштене процедуре и прикази који се односе на одређено груписање података сачуваних у табелама базе података.

Које типове табела чувамо у само једној бази података, а шта се налази у засебној бази података? Посматрајући једноставно рећи ћемо да се сви подаци за које се сматра да припадају само једном систему или да су значајно повезани, чувају у једној бази података. Систем *RDBMS* као што је *SQL* сервер може да има више корисничких база података на само једном серверу, а може да има и само једну. Број база података који може бити смештен на једном *SQL* серверу зависи од фактора као што су капацитет (снага *CPU, I/O* ограничења диска, меморија итд.), аутономија (желите да једна особа има право управљања сервером на којем се налази тај систем, а да неко други има администраторска права на другом серверу) или само број база података које ваша компанија или клијент има. Велики број сервера има само једну производну базу података, док их други могу имати и више. Треба имати на уму да код сваке верзије *SQL* сервера коју налазимо у употреби данас (*SQL* сервер 2000 је био већ пет година стар када је замењен), има могућност постојања већег броја примерака *SQL* сервера - укључујући и посебно пријављивање и посебна права управљања и то све на физички једном серверу.

Када се први пут покрене *SQL* сервер, почиње се са системским базама података:

- *мастер*
- *модел*
- *msdb*
- *tempdb*

2.4.2 Дневник трансакције

Сама датотека базе података није место на коме се дешава већина ствари. Иако се подаци дефинитивно читају одатле, свака измена која се прави не иде иницијално у саму базу података. Уместо тамо, они се серијски уписују у дневник трансакција. У неком каснијем тренутку база података задаје тренутак провере контролне тачке - у том тренутку времена све измене из дневника се преносе на саму датотеку базе података.

База података има уређење које се заснива на случајном приступу, али је сам дневник по својој природи серијски. Док случајан приступ датотеци базе података омогућује брз приступ, серијски приступ дневнику омогућује праћење ствари одговарајућим редоследом. Дневник акумулира измене за које се сматра да су урађене, а затим неколико измена одједном уписује у саму датотеку базе података. Да би имали функционалну базу података потребни су и датотека базе података и дневник трансакција.

2.4.3 Основни објекат базе података - табела

Базе података сачињавају многе ствари, али ни једна од њих није толико централна за саму садржину базе података као што је табела. Табела се може сматрати еквивалентом главној књизи рачуновође или радном листу у *Excelu*. Састоји се од такозваних података домена (колоне) и података ентитета (редови). Сами подаци из базе података чувају се у табелама.

Свака дефиниција табела садржи и метаподатке (описне информације о подацима) који описују природу података које би табела требало да садржи. Свака колона има свој скуп правила која одређују шта се може чувати у тој колони. Кршење правила у било којој колони може да доведе до тога да систем одбаци ред који желите да унесете или да одбије да ажурира постојећи ред или да спречи брисање реда.

2.4.4 Индекси

Индекс представља објекат који постоји само унутар радног окружења одређене табеле или приказа. Индекс функционише веома слично као и индекс на крају енциклопедије; постоји некаква вредност по којој се претражује (или кључна вредност), која је уређена на одређени начин и када се пронађе, добије се други кључ помоћу којег се могу пронаћи саме информације које су вам потребне.

Индекси обезбеђују начине који омогућавају да се убрза претраживање информација. Индекси спадају у две категорије:

Груписане - Само један овакав индекс може да постоји по табели. Ако је индекс груписани, то значи да је табела у којој се налази физички уређена у складу са тим индексом. Када би индексирали енциклопедију, кластерски индекс би био број стране; информације у енциклопедији су сачуване према редоследу бројева страна.

Негруписане – Може их бити више у свакој табели. Он се више уклапа у оно што се обично подразумева када се чује реч „индекс". Овај тип индекса указује на неку другу вредност која ће омогућити да се пронађу подаци. У енциклопедији, то би био индекс кључних речи на крају књиге.

Треба обратити пажњу на то да прикази који имају индексе или индексирани прикази - морају имати бар један груписани индекс да би уопште могли имати негруписане индексе.

2.4.5 Окидачи

Ограничење је још један објекат који постоји само унутар граница табеле. Ограничења су баш то што и њихово име говори, објекти који ограничавају податке у табели тако да испуњавају одређене критеријуме. Ограничења се на неки начин такмиче са окидачима као могућа решења за захтеве у вези са интегритетом података. Међутим, они не представљају исту ствар јер имају различите предности.

2.4.6 Групне датотеке

Према подразумеваној опцији, све табеле и све остало у вези са базом података (изузев дневника) чува се у једној датотеци. Та датотека је део нечега што се назива примарна група датотека. Међутим, то није све што се тиче уређења.

SQL сервер дозвољава дефинисање нешто више од 32000 секундарних датотека. Те секундарне датотеке могу се придружити примарној групи датотека или се могу направити као део једне секундарне групе датотека или више таквих група. Док постоји само једна примарна група датотека (и она се у свари назива „*Primary*"), могу постојати до 255 секундарних група датотека. Секундарна група датотека се прави као опција у оквиру команде *CREATE DATABASE* или *ALTER DATABASE*.

2.4.7 Дијаграми

Дијаграм базе података визуелно представља дизајн базе података укључујући различите табеле, имена колона у свакој табели и везе између табела. Постоји термин дијаграм ентитет-веза или *ERD*. На *ERD* дијаграму база података је подељена на два дела: ентитета (као што су „снабдевач" и „производ") и везе (као што су „набавка" и „куповина").

2.4.8 Прикази

Приказ је нешто налик на виртуелну табелу. У већини случајева, приказ се и користи као табела, али се од табеле разликује по томе што не садржи своје податке. Приказ представља унапред планирано мапирање и представљање података који се чувају у табелама. Тај план се чува у бази података у облику упита. Упит захтева податке из неколико колона (није неопходно да буду из свих) из једне или више табела. Враћени подаци можда морају, или не (зависно од дефиниције приказа) да испуњавају одређене критеријуме да би се појавили у том приказу. До *SQL* сервера 2000 основна сврха приказа била је контрола онога што корисник може да види. На то утичу две основне ствари: безбедност и лакоћа коришћења. Са приказима можете да контролишете оно што корисници виде, тако да ако постоји део табеле којем може да приступи само неколико корисника (на пример, детаљи у вези са платама), можете да направите приказ који садржи само колоне којима сви могу да приступе. Поред тога, приказ се може направити тако да корисник не мора да претражује скроз непотребне информације. Поред ових најосновнијих примена приказа, постоји могућност да се направи и такозвани индексирани приказ. Он је исти као и било који други приказ, изузев што у односу на тај приказ можемо да направимо индекс.

То има неколико утицаја на перформансе (неки су позитивни, а неки негативни):

- Прикази који референцирају више табела извршавају се много брже ако је приказ индексиран зато што је спајање табела унапред направљено.
- Агрегирања која се обављају у приказу унапред се рачунају и чувају се као део индекса, то опет значи да се агрегација обавља у једном тренутку (када се унесе или ажурира неки ред), а затим се може читати из информација које садржи индекс.
- Уношење или брисање реда је спорије зато што и индекс у приказу мора одмах да се ажурира, ажурирања су такође спорија ако ажурирање утиче на колону која представља кључ у индексу.

2.4.9 Ускладиштене процедуре

Ускладиштене процедуре су историјски, па чак и у ери *.NET*-а, основа програмске функционалности код *SQL* сервера. Ускладиштене процедуре у ствари представљају низове исказа *Transact-SQL* (језик који се користи за задавање упита код *Microsoft SQL* сервера) спојених у једну логичку целину. Дозвољавају коришћење променљивих и параметара као и конструкција које омогућују избор и пролазак у циклусима. Ускладиштене процедуре нуде неколико предности у односу на слање појединачних исказа серверу због следећих разлога: Референцирају се коришћењем кратких имена, уместо дугачких низова текста, због чега је потребно мање мрежног саобраћаја како би се извршио код унутар ускладиштене процедуре. Оне су преоптимизоване и прекомпајлиране, чиме се штеди мала количина времена сваки пут када се ускладиштена процедура извршава. Учаурују процес, обично из безбедносних разлога или да би се сакрила сложеност базе података. Могу се позивати из других ускладиштених процедура, због чега се могу сматрати поново употребљивим у неком ужем смислу. Поред тога, можете да користите било који *.NET* језик ако у ускладиштене процедуре желите да убаците програмске конструкције које се не налазе у матерњем језику *T-SQL*.

2.4.10 Функције које дефинише корисник

Функције које дефинише корисник (*UDF*-ови) имају веома много сличности са ускладиштеним процедурама, а разликују се по томе што:

Могу да врате вредност чији тип може бити већина типова података који постоје код *SQL* сервера. Повратна вредност не може бити типа *text, image, cursor, timestamp*. Не могу имати „споредне ефекте". У основи, оне не могу даураде ништа изван домена функције, као што је мењање табела, слање електронских порука или мењање параметара система или параметара базе података.

Функције које дефинише корисник сличне су функцијама које бисте користили у стандардном програмском језику као што је *BC.NET* или *C++*. Можете да проследите више од једне променљиве и да добијете резултат на основу њих. Функције које дефинише корисник код *SQL* сервера разликују се од функција које можете наћи у многим процедуралним језицима али по томе да се све променљиве које се прослеђују функцији увек прослеђују по вредности. Међутим, постоје неке добре вести, а то су да можете вратити посебан тип података који представља табелу.

2.4.11 Корисници и роле

Ово двоје увек иду заједно. Корисници су прилично еквивалентни налозима. Укратко, овај објекат представља идентификатор потребан да би се неко пријавио на *SQL* сервер. Свако ко се пријављује на *SQL* сервер мора да се преслика у корисника (директно или индиректно зависно од безбедносног модела који се користи). Корисници припадају једној роли или већем броју рола. Права да се обаве одређени поступци код *SQL* сервера могу се доделити директно кориснику или роли којој припада један или више корисника.

2.4.12 Правила

Правила и ограничења ограничавају информације које могу да уђу у табелу. Ако ажурирани или убачени запис крши правило, убацивање или ажурирање ће се одбацити. Поред тога, правило може да се користи за дефинисање ограничења код кориснички дефинисаних типова података. За разлику од правила, ограничења нису баш самостални објекти, већ делови метаподатака који описују одређену табелу.

Правила се морају сматрати објектом који ту постоји само због компатибилности уназад и требало би их избегавати у новим апликацијама.

2.4.13 Подразумеване вредности

Постоје два типа подразумеваних вредности. Постоји подразумевана вредност која представља самосталан објекат и подразумевана вредност која у ствари и није објекат већ представља метаподатке који описују одређену колону у табели (слично као што имамо правила, која представљају објекте, и ограничења, која нису објекти већ метаподаци). Имају исту намену. Ако приликом убацивања новог реда не обезбедите вредност за неку колону, а колона има дефинисану подразумевану вредност, аутоматски ће се убацити вредност која је дефинисана као подразумевана.

2.4.14 Типови података које дефинише корисник

Типови података које дефинише корисник представљају проширења за типове података које дефинише систем. Почевши од ове верзије *SQL* сервера, могућности у овој области су скоро бескрајне. Иако су *SQL* сервер 2000 и претходне верзије имали појам типова података које дефинише корисник, ти типови података били су ограничени на различито филтрирање постојећих типова података. Код *SQL* сервера 2005 имате могућност да везујете *.NET* склопове за ваше типове података, што значи да можете имати тип података који чува (са разлогом) све што можете сачувати у *.NET* објекту.

2.4.15 Подаци типа *NULL*

Шта ако имате ред који не садржи никакве податке у одређеној колони, то јест, шта ако једноставно не знате ту вредност? На пример, рецимо да постоји запис који покушава да чува информације о раду компаније током дате године. Сада замислите да је једно од поља проценат пораста у односу на претходну годину, али немате записе за годину пре првог записа у бази података. Можете доћи у искушење да унесете нуле у колону. Међутим, да ли ће то обезбедити праве информације? Они који не знају детаље могу помислити да та нула означава да сте имали проценат раста од нула посто, а не чињеницу да једноставно не знате вредност за посматрану годину.

Вредности које су неодређене означавају се као вредности *NULL*. Веома је тешко дефинисати вредност *NULL* јер значи да не знате која је вредност. Могла би бити 1; могла би бити 347; могла би бити -294 и то је све што знамо. Укратко, значи да је вредност недефинисана или можда неприменљива.

3. Основне наредбе *SQL*-а

Основна наредба у *SQL*-у је *SELECT* израз *FROM* име таблице, у преводу ИЗАБЕРИ израз ИЗ име таблице. У наредби реч ИЗРАЗ замењује се са именима редова које је потребно приказати или у случају да је потребно видети све са *. Поред имена реда може се написати ново име реда под којим ће се видети у извештају, али у таблици остаје све по старом.

Следећа *SELECT* наредба се разликује код *SQL*-а и *SAP*-а. Као што је приказано у претходној наредби да сваки ред има своје име, али некад може бити потребно да се у неком извештају споје два реда таблице у један ред извештаја. Спајање се врши у *SQL* -у са знаком +, док се иста радња у *SAP*-у ради са знаком назива пипе (*Alt Gr + W*). Можемо још споменути и кључну реч *TOP x*, која нам даје само првих *x* редова, што је корисно при раду са великим таблицама када често морамо проверавати резултате наших израза. Постоји још једна могућност коришћења наредбе *SELECT* без додатка *FROM* јер тај податак не мора бити унешен у таблицу. Следећи користан додатак наредби *SELECT* је захтев *WHERE* који показује који се редови могу видети јер у већини претходних примера имамо приказане све редове.

Наредни корак уношења података би био употреба логичких оператора тако да је могуће још више проширити захтеве за претраживање. Логички оператори су *AND, OR* и *NOT*. Код сложенијих израза би требало водити рачуна о приоритетима. Хијерархија примене израза гласи:

- Заграда ( ),
- Дељење / и множење *,
- Сабирање + и одузимање -,
- *NOT* (не),
- *AND* (и),
- *OR* (или).

Ако је потребно дефинисати у захтеву распон имамо две могућности. Једна је да радимо са операторима упоређивања (<, =, >, ...), а друга је да користимо кључну реч *BETWEEN*. Међутим, уколико је потребно наћи податак којем је захтев да се поклапа са једном од вредности у листи користимо кључну реч *IN*. Некад се може догодити да тражимо, на пример, неко име које почиње као ... или телефонски број. Тако задате захтеве је мало теже добити кроз прошле наредбе или операторе, па за то служи кључна реч *LIKE*. Оператор *IS NULL* нам даје информацију, на пример, за коју особу из таблице немамо податке. Уз коришћење ове кључне речи може се добити резултат који је читљивији, односно можемо резултат сортирати по неком редоследу.

Међутим, кад постоје само бројеви или само слова јасно је како ће бити сортирани ти низови, али кад је све то измешано поставља се питање, како ће сад то бити сортирано? Ту можемо позвати у помоћ једну процедуру у *MS SQL*-у која ће нам дати редослед низа по којем се врши сортирање, а то је *EXEC SP_HELPSORT*.

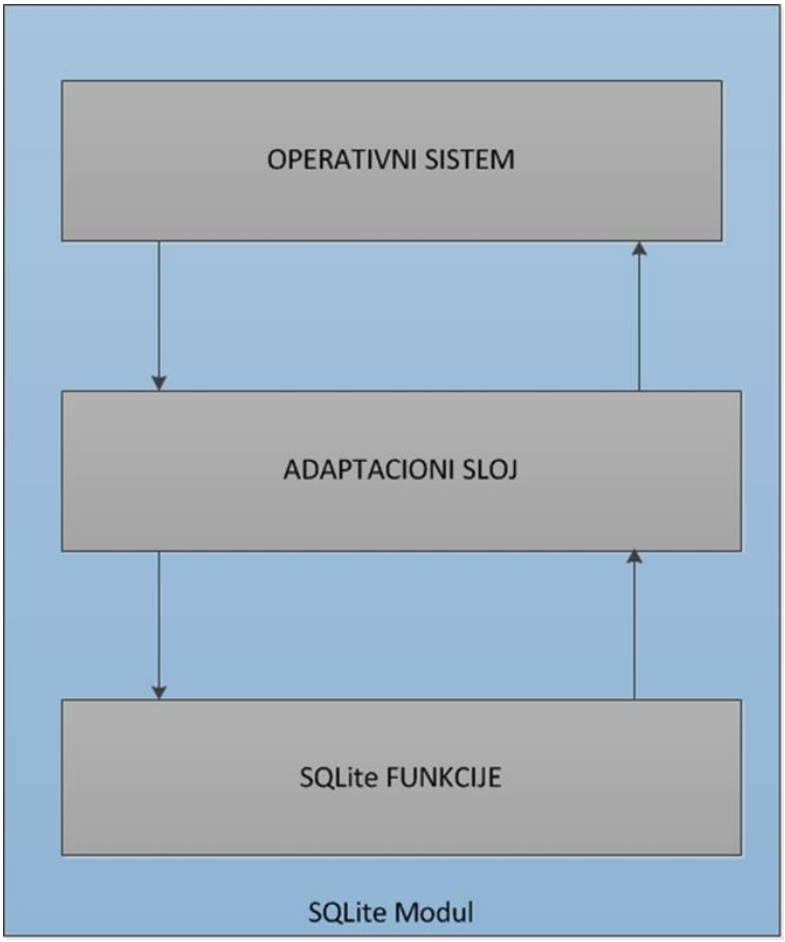
4. *SQLITE*

SQLite је софтверска библиотека написана у *C* програмском језику која имплементира *SQL* машину базу података. *SQL* је програмски језик намењен за управљање подацима у релационом систему за управљање базама података. Обухвата унос података, ажурирање, брисање, упите као и податке за контролу приступа. Одлика *SQLite* јесте да не захтева велике перформансе, брза је , мала и портабилна. Процеси читања и писања података се врше директно у фајл базе података. Нису потребна додатна подешавања, већ само укључивање модула у рад програма.

SQLite је уграђен у све већи број популарних програма и оперативних система. На пример *Mozilla Firefox* чува мноштво конфигурационих података у интерно управљану *SQLite* базу података. Исто тако и *Android* оперативни систем за мобилне телефоне и друге мале уређаје садржи *SQLite*.

У зависности од циљне платформе и оперативног система који се користи, често се разликује рад са системским позивима задуженим за руковање меморијом. У случају промене платформе, преласка на други оперативни систем или интеграције *SQLite* модула за неке друге сврхе, направљен је један адаптациони слој преко којег функционишу позиви према флеш меморији као и системски позиви за функције које су специфичне за циљну платформу на којој се врши интеграција *SQLite* библиотеке.

Сврха адаптационог слоја јесте да позиве упућене ка меморији прилагоди облику који захтева платформа, а повратне вредности функција (у зависности од платформе) прилагоди захтевима *SQLite* библиотеке. На следећој слици је приказан адаптациони слој између *SQLite* библиотеке и оперативног система.



Слика 1. Адаптациони слој [1]

Међутим, због тренутне конфигурације оперативног система на којем ради уређај за репродукцију дигиталног телевизијског сигнала као и његова максимална оптимизација по питању перформанси, неке системске функције које користи *SQLite* библиотека недостају. Мимо тих функција које недостају, постоје и функције са истим називом, али разлике су у улазним аргументима као и повратне вредности код неких функција. То је један од разлога за постојање адаптационог слоја који раздваја те функције, које су специфичне за овај оперативни систем који користи *SQLite* библиотеку. У сврху коришћења овог модула на неку другу платформу која користи други начин чувања података на флеш или масовној меморији, потребне су измене тела функција које се налазе у поменутом адаптационом слоју.

5. Складиште података

Складиште података је скуп података на којем се темељи систем подршке у одлучивању. Из сврхе произилази да складиште података треба да подацима потпуно "покрије" једно или више пословних подручја (нпр. набавке, продаје), да подаци у складишту требају да буду свеобухватни. Подаци морају да обухвате дужи временски период (пет, десет или више година), јер су временске анализе пословно врло значајне. Оријентисаност према пословним анализама не захтева од складишта да се подаци брзо ажурирају као у бази података. Мање складиште података које обухвата податке само једног пословног подручја назива се подручним складиштем података (*Data mart*).

Складиште података намењено је менаџерима, али и свима који у свом послу обављају различите аналитичке задатке, као што су послови праћења и извештавања, који се темеље на примени различитих пословних правила.

Подаци у складишту су стабилни и кад се једном сместе у складишту по правилу се не мењају. Тиме се омогућује да менаџемент или свако ко користи складиште података може бити сигуран да ће добити једнак одговор независно од времена или учесталости постављања упита. Поступак складиштења података представља континуалан процес планирања, грађења и прикупљања података из различитих извора, његовог коришћења, одржавања, управљања и сталног унапређења. Међу многим корацима у том комплексном континуалном процесу битно је нагласити важност поседовања визије о томе што се жели постићи креирањем складишта података. Једна од улога складишта је пример развоја и коришћење знања заснованог на подацима *(data-based knowledge).*

Главни циљ складишта података је ослободити информације које су "закључане" у базама података и "помешати" их са информацијама из осталих, по правилу спољних извора података. Велике организације данас све више траже додатне податке из спољашњих извора, као што су на пример подаци о конкуренцији, демографски трендови, продајни трендови и сл.

Да би складиште података могло да испуни циљ и сврху свог постојања, мора пре свега да испуни одређене предуслове:

1. Мора осигуравати приступ свим запосленим у предузећу, а не само менаџерима, значи може служити великом броју људи. Тај приступ мора бити поуздан, брз и једноставан.
2. Складиште треба садржати велику количину детаљних података. То значи да све пословне трансакције релевантне за доношење пословних одлука, које су настале у процесима предузећа морају бити евидентиране у складишту података. Унесени подаци требају бити конзистентни, нпр. ако је са два различита места у различито време постављен једнак упит и резултат тих упита мора бити исти.
3. Освежавање, односно ажурирање новим подацима треба бити континуалан процес, по могућности треба се одвијати у стварном времену практично одмах након што се неки пословни догађај одиграо или одмах по завршетку неког процеса.
4. Мора бити увек расположиво и обликовано на начин да може послужити свакој ситуацији коју није увек могуће унапред предвидети.
5. Треба предвидети могућност издвајања и међусобног повезивања података у смислу добијања свих мера и показатеља пословања у подузећу.

6. Подаци у складишту који се скупљају из различитих извора, контролишу се уз осигурање квалитета и само такви су доступни корисницима. Лоши улазни подаци не могу давати добре излазне податке.
7. Мора бити проширив да би могло пратити стратегију проширења пословања предузећа.
8. И на крају, мора да задовољи одговарајуће мере заштите тајности осетљивих података што се постиже спровођењем ригорозних мера чувања тајности.

6. Коришћена радна окружења

6.1 *Firebird* база података

Firebird SQL језик је свеобухватни приручник који покрива све аспекте језика упита који програмери користе за комуникацију и интеракцију са базом, путем својих апликација, са *Firebird*-овим системом управљања релационим базама података. Има дугу историју. Предмет овог дела је у потпуности *Firebird*-ова имплементација *SQL* релације језика базе података. *Firebird* се блиско усклађује са међународним стандардима за *SQL*, од типа података подршка, структуре за складиштење података, референтни механизми интегритета, за манипулацију подацима могућности и привилегије приступа. Такође примењује робустан процедурални поступак, језик процедурални *SQL(PSQL)* за ускладиштене процедуре, окидаче и динамички извршну датотеку кодних блокови.

Изразити подскупови *SQL*-а примењују се на различите секторе активности. Подскупови *SQL* на језику *Firebird* имплементације су:

• Динамички *SQL*

• Процедурални *SQL*

• Уграђени *SQL*

• Интерактивни *SQL*

Динамички *SQL* је главни део језика и представља изразе које су клијентске апликације проследиле путем јавног *Firebird API*-а и обрађује га механизам базе података.

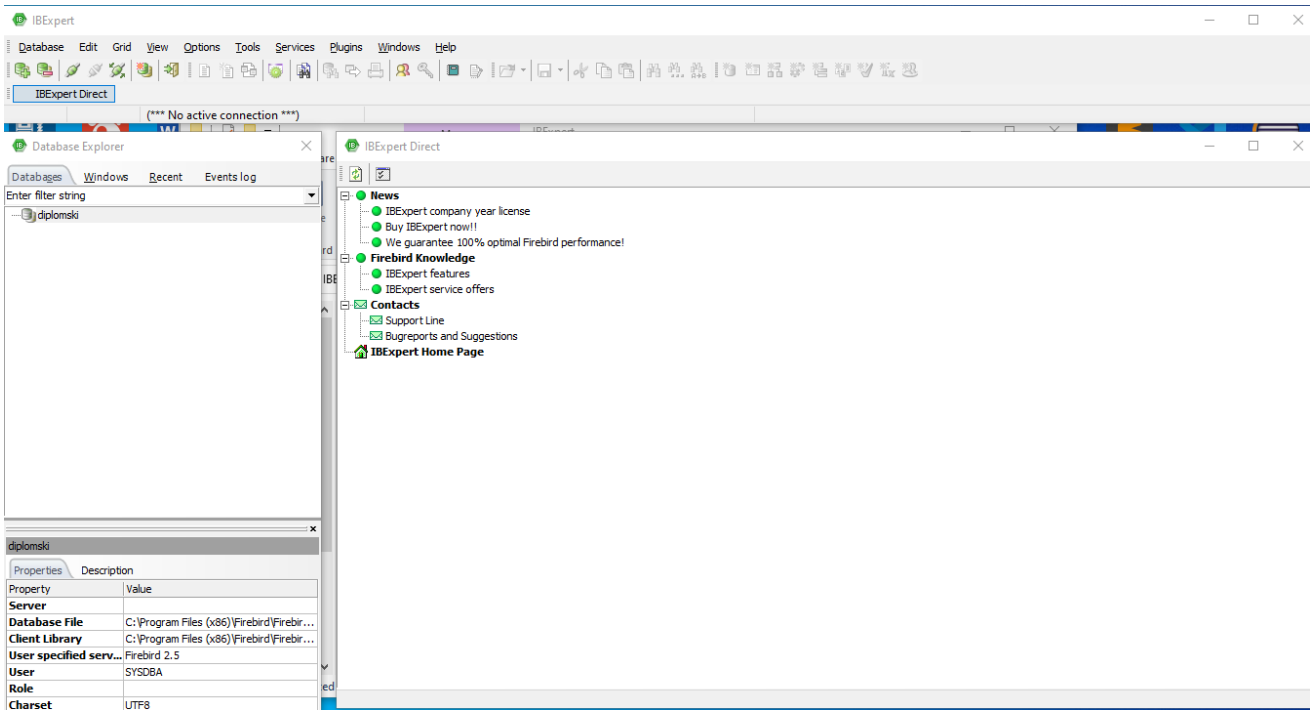
Процедурални *SQL* увећава динамички *SQL* како би омогућио сложене изразе који садрже локалне променљиве, задаци, услови, петље и друге процедуралне конструкције.

Уграђени *SQL* дефинише динамички *SQL* подскуп који подржава *Firebird gpre*, апликација која омогућава уградњу *SQL* конструкције у програмски језик (*C, C ++, Pascal*) унапред обрађује те уграђене конструкције у одговарајуће *Firebird API* позиве.

Интерактивни *SQL* односи се на језик који се може извршити помоћу *Firebird isql*, командне линије, апликација за интерактивни приступ базама података. Као уобичајена клијентска апликација, њен матерњи језик је *DSQL*. Такође нуди неколико додатних команди које нису доступне изван његове специфичности.

6.2 *IBExpert*

IBExpert (Слика 2) је професионално интегрисано развојно окружење (*IDE*) за развој и управљање базама података *InterBase* и *Firebird*.



Слика 2. Изглед IBExpert

Потребно је инсталирати *FireBird* (исталациони фајл пронаћи на интернету)

Приликом инсталације све прихватити по *default*-у, осим ако је процесор са два језгра и више чекирати - *Classic* у супротоном, чекирати - *Super server binary*

За опцију – *Sellect additional task* - чекирати – *Copy firebird library to system directory*.

Након инсталације неопходно је подесити алијас *Firebird*-а (Све ово ради се на рачунару на ком треба сместити базу)

Са десне стране уписујемо назив базе, док са леве стране уписујемо локацију где је она смештена.

Јако је битно нагласити да се као коментар увек пише назив базе и датум подизања.

ФБК- је екстензија за *BackUp*

ФДБ – је екстензија за *Restore*.

Уколико се после *_EX* нађе број значи да је стара база прегажена новим *Restor*-ом

6.3 *.NET*

Окружење за развој софвера, развијано од стране *Microsoft*-а за *Windows* платформе, *iOS* , *Android OS*. Укључује велику библиотеку класа *(Framework Class Library). Microsoft* је са развојем *.NET*-а почео раних 1990-тих, под називом *Next Generatio Windows Services***.** Почетком 2000-тих прва бета верзија *.NET* 1.0 је објављена, а у августу 2000. у сарадњи са *Intel*-ом и *HP*-ом, *Microsoft* је почео са стандардизацијом *Common Language Infrastrukture (CLI),* која ће омогућити извршавање различитих програмских језика на различитим архитектурама-платформама.

Технологија *Microsoft .NET Framework*-а за развој динамичих веб сајтова, интерактивних веб апликација и веб сервиса са коришћењем базе података за *PC* и мобилне уређаје. *APS.NET* странице се извршавају на серверској страни и генеришу *HTML, WML* или *XML* који се шаље десктоп или мобил претраживачима. *ASP.NET* користи "*event-driven*" модел програмирања који побољшава перформансе и омогућава сепарацију корисничког интерфејса од логике апликације. *ASP.NET* ради на врху *HTTP* протокола користећи *HTTP* команде и правила како би омогућио обострану комуникацију између клијента и сервера. Код је могуће писати у *C#, VisusalBasic* и *Jscript*-у. У овом раду кориштен је програмски језик *C#* у *Visual Studio 2019* окружењу. *ASP.NET* омогућава 3 методе развоја:

1. *Web Forms*
2. *Web Pages*
3. *MVC (Model View Controller)*

Web Forms омогућавају сепарацију *HTML*-а и осталог *UserInterface* кода од логике апликације, приступ подацима, моћни дата *binding*, подршку за *skripting* са клијентске стране као и остале могућности као што су рутирање, безбедност... Најстарији је *ASP* програмски модел са *„event-driven"* веб странама писаних у комбинацији *HTML*-а, серверских контрола и серверског кода. *Web Pages* је најједноставнији метод, *HTML* и код се налазе у истом фајлу, лак је за учење и сличан *PHP*-у.

MVC (Model View Controller) омогућава моћан, *pattern*-оријентисан развој у 3 слоја:

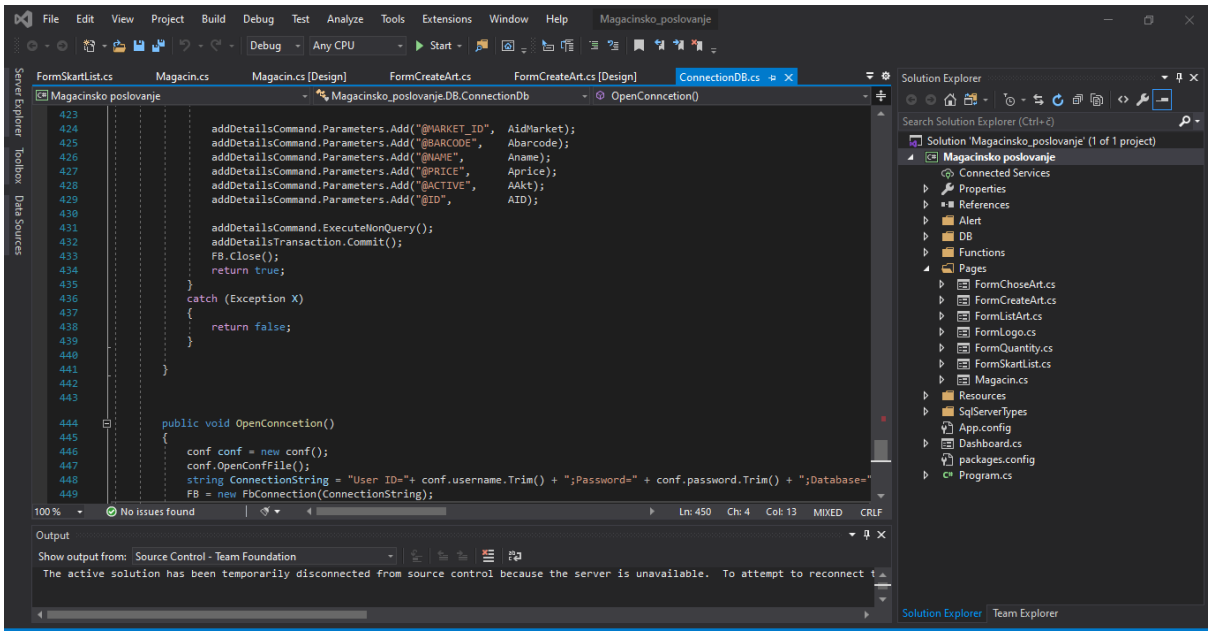
Model - на овом слоју је бизнис логика, начешће за комуникацију са базом,

View - омогућава приказ и најчешће приказује податке из модела,

Controller - је део апликације који чита са *View*-а, контролише корисничке *input*-е, и прослеђује *input* податке *Model*-у.

6.4 *Microsoft Visual Studio 2019*

Microsoft Visual Studio је интегрирано развојно окружење (*IDE*) које се користи за развој рачунарских програма за *Windows*, веб странице, апликације и друге услуге. Користи *Microsoft*-ове платформе за развој попут апликативних програмских интерфејса (*API*) за *Windows, Windows Forms, Windows Presentation Foundation, Windows Store и Microsoft Silverlight.*



Слика 3. Изглед Microsoft Visual Studia

Visual Studio садржи уређивач изворног кода који подржава *IntelliSense* (компонента која предлаже остатак кода) као и рефакторирање кода. Интегрирани програм за отклањање грешака (*debugger*) ради на нивоу изворног кода и машинског кода. Програм такође садржи алате попут дизајнера облика који се користи за прављење апликација са графичком корисничким интерфејсом, веб-дизајнера, дизајнера класа и дизајнера шеме базе података. Прихвата проширења која побољшавају функционалност на скоро сваком нивоу додајући подршку система за управљање изворним кодом и додајући нове низове алата попут текстуалних уређивача и визуалних дизајнера за језик одређених домена или за друге делове процеса развоја софтвера. Софтвер не подржава верзије оперативних система ниже од *Windows 7.*

Предности:

- Брза и једноставна инсталација на лични рачунар
- Општеприхваћени стандард за програмирање у оперативном систему *Windows*
- Подршка за велики број језика и формата
- Брза интерпретација кода
- Лагана имплементација веб апликација
- Интегрирана подршка за надоградњу базе кода

Недостатци:

- Уређивач можда није погодан за почетнике
- Учитава рачунарски систем

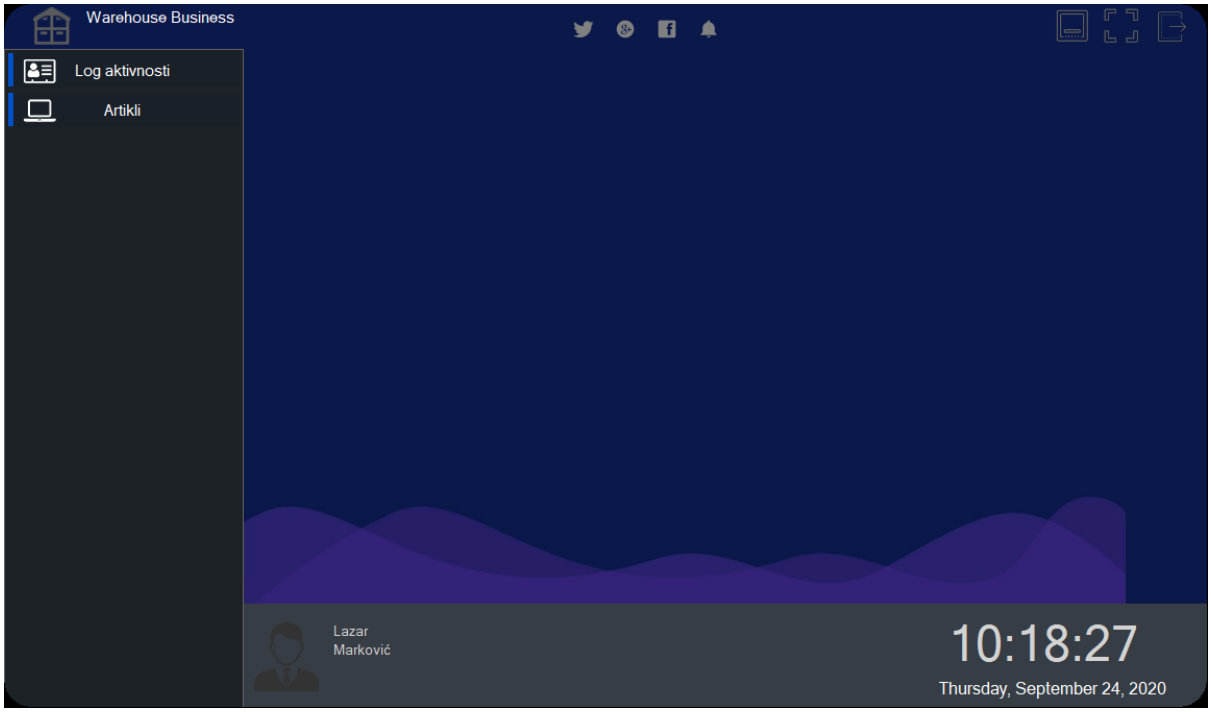
Кључне карактеристике :

- Стварање апликација за разне платформе и оперативне системе
- Прикладан уређивач кода опремљен додатним "помагачима"
- Доступност алата за преуређивање кода
- Подршка за популарне програмске језике *Python, C++* и друге
- Приступ 5000 проширења *Visual Studio*-а
- Програм је доступан потпуно бесплатно
- Могу се користити у комерцијалне и некомерцијалне сврхе

7. Опис апликације

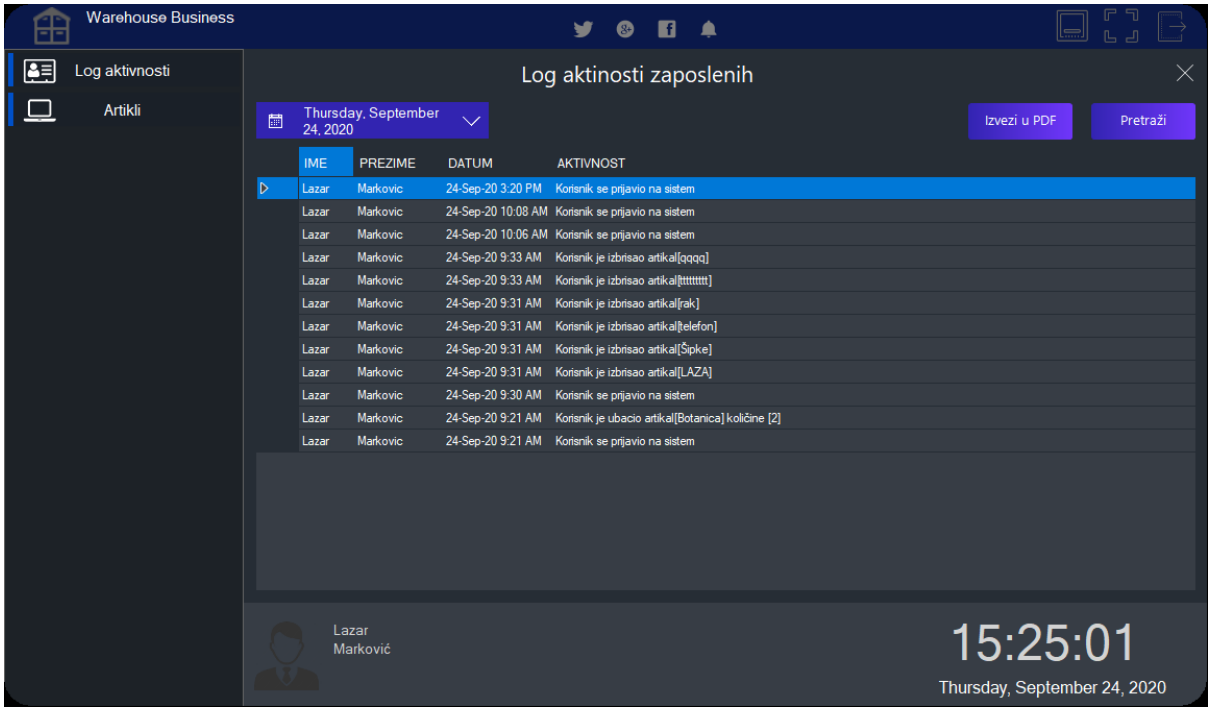
Идеја ове апликације је да се олакша вођење евиденције при трговини и складиштењу робе. При покретању апликације корисник може погледати који се артикли налазе у магацину, која је количина којом располаже, цене артикла и додавање нових артикала. Поред тога може погледати све активности које су обављене са овим системом. У наставку следи упутство коришћења.

При покретању апликације корисник добија почетни изглед са две опције *"Log aktivnosti"* и *"Artikli"* који је приказан на слици 4.



Слика 4. Почетна страна апликације

У колико корисник одабере опцију *"Log aktivnosti"* отвара му се евиденција као на слици 5. Овде корисник има две опције : да извезе пдф или погледа за одговарајући датум шта је додато у базу, измењено или обрисано. Како би погледао за одговарајући датум, потребно је да означи датум и кликне на дугме *"Pretraži"* након чега добија излистану базу података са евиденцијом за тај датум.

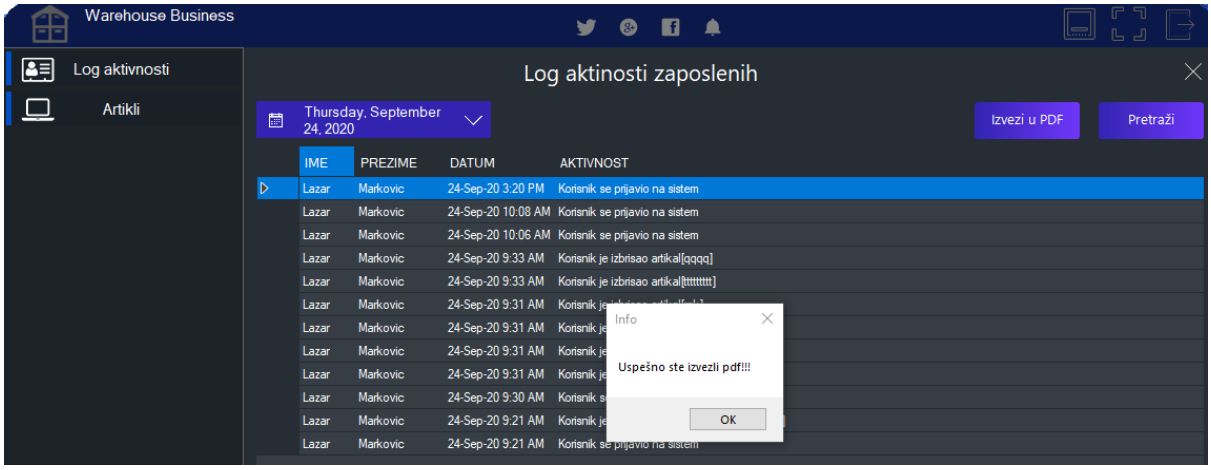


Слика 5. Евиденција активности

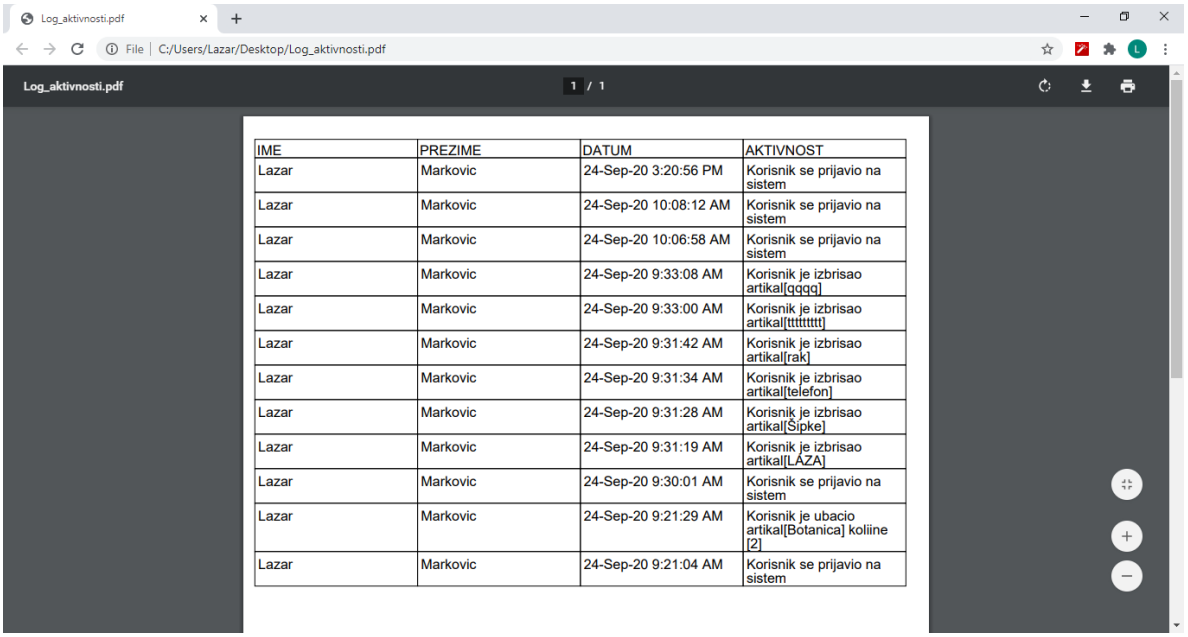
```
SET SQL DIALECT 3;
/******************************************************************************/
/*                                  Tables                                    */
/******************************************************************************/
CREATE GENERATOR GEN_LOG_ACTIVITY_ID;
CREATE TABLE LOG_ACTIVITY (
    ID         INTEGER NOT NULL,
    USER_ID    INTEGER NOT NULL,
    DAT        TIMESTAMP NOT NULL,
    DESC_USER  VARCHAR(255) CHARACTER SET UTF8
);
/******************************************************************************/
/*                                Primary keys                                */
/******************************************************************************/
ALTER TABLE LOG_ACTIVITY ADD PRIMARY KEY (ID);
/******************************************************************************/
/*                                Foreign keys                                */
/******************************************************************************/
ALTER TABLE LOG_ACTIVITY ADD CONSTRAINT FK_LOG_ACTIVITY_1 FOREIGN KEY (USER_ID) REFERENCES USERS (ID);
/******************************************************************************/
/*                                  Triggers                                  */
/******************************************************************************/
SET TERM ^ ;
/******************************************************************************/
/*                            Triggers for tables                             */
/******************************************************************************/
/* Trigger: LOG_ACTIVITY_BI0 */
CREATE OR ALTER TRIGGER LOG_ACTIVITY_BI0 FOR LOG_ACTIVITY
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.ID IS NULL) THEN NEW.ID = GEN_ID(GEN_LOG_ACTIVITY_ID, 1);
END
^
```

Слика 6. SQL код за креирање табеле log_activity

Ако се одлучи за пдф, потребно да је кликне на дугме *"Izvezi u PDF"*, а затим је потребно да одабере локацију где ће се сачувати документ. Ако је успешно извезао пдф, добија обавештење као на слици 7. Изглед сачуваног документа приказан је на слици 8.



Слика 7. Обавештење за пдф



IME	PREZIME	DATUM	AKTIVNOST
Lazar	Markovic	24-Sep-20 3:20:56 PM	Korisnik se prijavio na sistem
Lazar	Markovic	24-Sep-20 10:08:12 AM	Korisnik se prijavio na sistem
Lazar	Markovic	24-Sep-20 10:06:58 AM	Korisnik se prijavio na sistem
Lazar	Markovic	24-Sep-20 9:33:08 AM	Korisnik je izbrisao artikal[qqqq]
Lazar	Markovic	24-Sep-20 9:33:00 AM	Korisnik je izbrisao artikal[tttttttt]
Lazar	Markovic	24-Sep-20 9:31:42 AM	Korisnik je izbrisao artikal[rak]
Lazar	Markovic	24-Sep-20 9:31:34 AM	Korisnik je izbrisao artikal[telefon]
Lazar	Markovic	24-Sep-20 9:31:28 AM	Korisnik je izbrisao artikal[Šipke]
Lazar	Markovic	24-Sep-20 9:31:19 AM	Korisnik je izbrisao artikal[LAZA]
Lazar	Markovic	24-Sep-20 9:30:01 AM	Korisnik se prijavio na sistem
Lazar	Markovic	24-Sep-20 9:21:29 AM	Korisnik je ubacio artikal[Botanica] koliine [2]
Lazar	Markovic	24-Sep-20 9:21:04 AM	Korisnik se prijavio na sistem

Слика 8. Изглед пдф-а

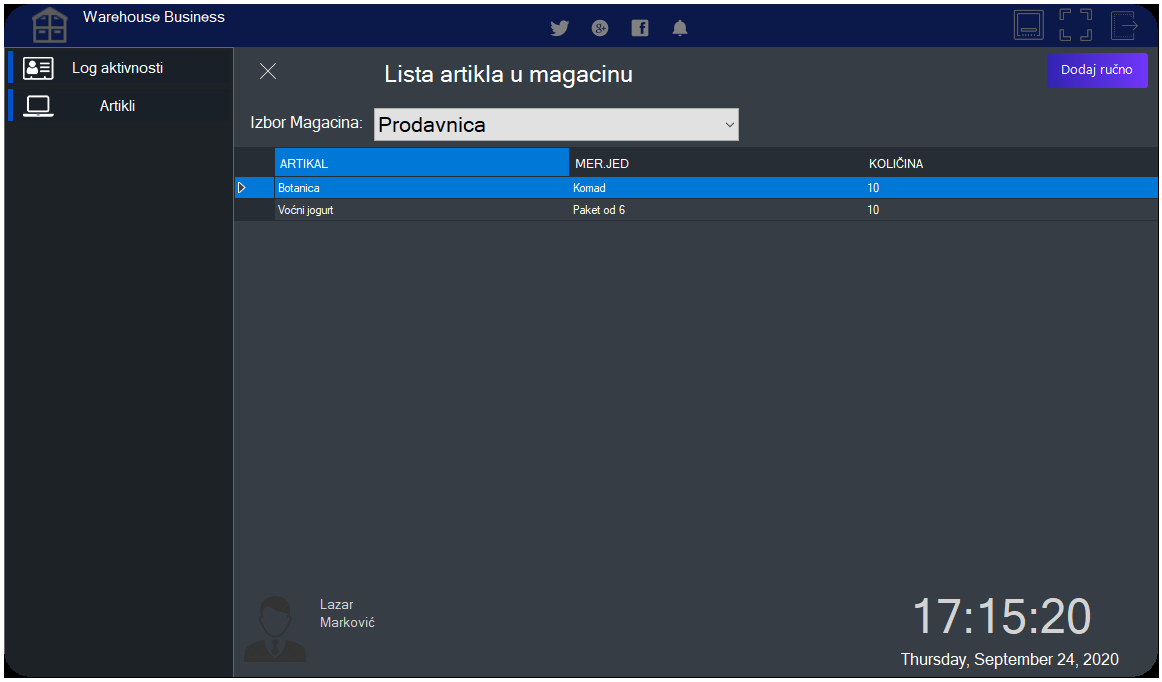
Ако корисник на почетној страни одабере опцију *"Artikli"* отвара му се падајући мени где може да изабере:

1. *"Magacin"*
2. *"Novi artikal"*
3. *"Škart"*

Ако изабере *"Magacin"* отвара му се листа артикла у магацину. Ту постоји могућност да се погледају сви артикли који постоје у магацину или да се изабере конкретан магацин који жели да погледа. Приказ је дат на слици 10. Изглед *sql* кода дат је на слици 9.

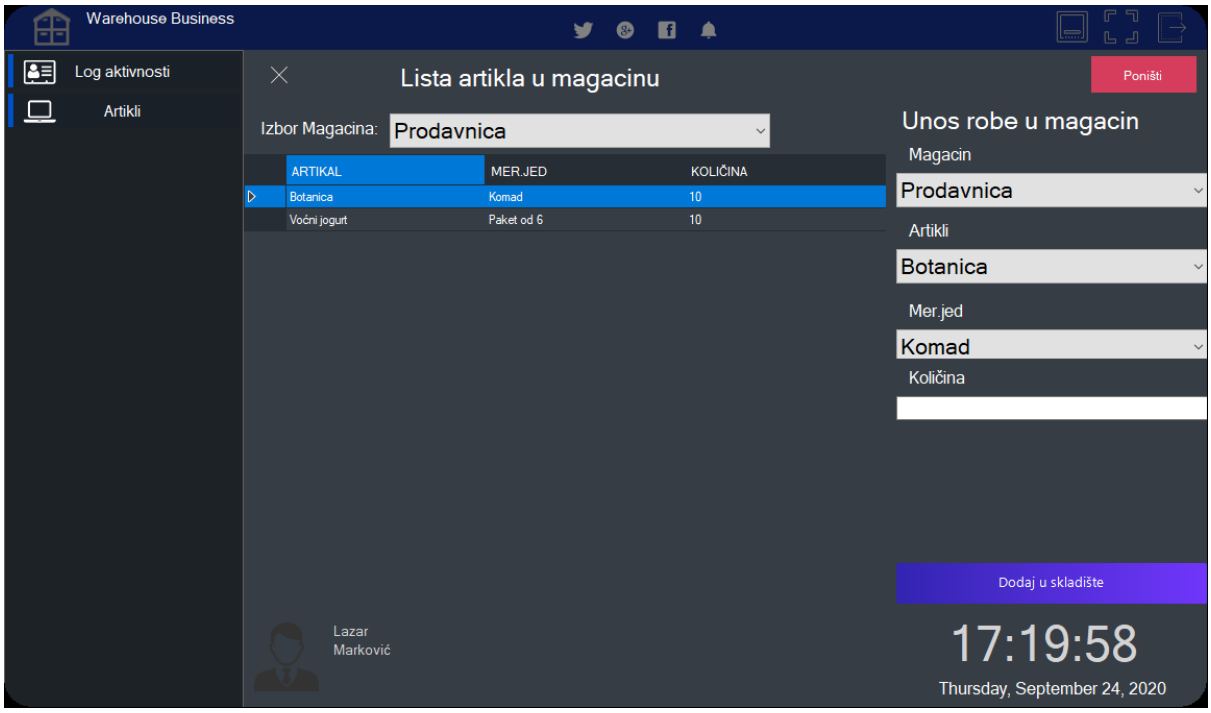
```
SET SQL DIALECT 3;
CREATE GENERATOR GEN_MARKET_PLACE_ID;
CREATE TABLE MARKET_PLACE (
    ID        INTEGER NOT NULL,
    NAME      VARCHAR(50) CHARACTER SET UTF8,
    "ACTIVE"  INTEGER DEFAULT 0 NOT NULL,
    ADDRESS   VARCHAR(50) CHARACTER SET UTF8
);
ALTER TABLE MARKET_PLACE ADD PRIMARY KEY (ID);
SET TERM ^ ;

/* Trigger: MARKET_PLACE_BI0 */
CREATE OR ALTER TRIGGER MARKET_PLACE_BI0 FOR MARKET_PLACE
ACTIVE BEFORE INSERT POSITION 0
AS
begin
    IF (NEW.ID IS NULL) THEN NEW.ID = GEN_ID(gen_market_place_id, 1);
end
^
SET TERM ; ^
COMMENT ON COLUMN MARKET_PLACE.NAME IS
'Ime mesta gde skladisti robu';
COMMENT ON COLUMN MARKET_PLACE."ACTIVE" IS
'AKO JE 0 ONDA JE AKTIVAN AKO JE 1 ONDA NIJE';
/* Trigger: LOG_ACTIVITY_DAT */
CREATE OR ALTER TRIGGER LOG_ACTIVITY_DAT FOR LOG_ACTIVITY
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  NEW.DAT = CURRENT_TIMESTAMP;
END
^
SET TERM ; ^

COMMENT ON COLUMN LOG_ACTIVITY.DESC_USER IS
'Opis sta je taj kornsik uradio';
```

Слика 9. *SQL код за креирање табеле market_place*



Слика 10. Магацин

Осим овог приказа имамо могућност одабира опције *"Dodaj ručno"* чији је приказ дат на слици 11. Приликом уноса робе у магацин потребно је одабрати у који магацин желимо да додамо артикал, након изабраног магацина и кликом на друго поље везано за *"Artikli"* одаберe артикал који желимо у одабраном магацину, затим мерну јединицу (комад, литар, паковање од 20 или 6) и на крају количину. *SQL* код који регулише одабир мерне јединици дат је на слици 12. Кликом на *"Dodaj u skladište"* у базу се уноси назив артикла са мерном јединицом и количином.

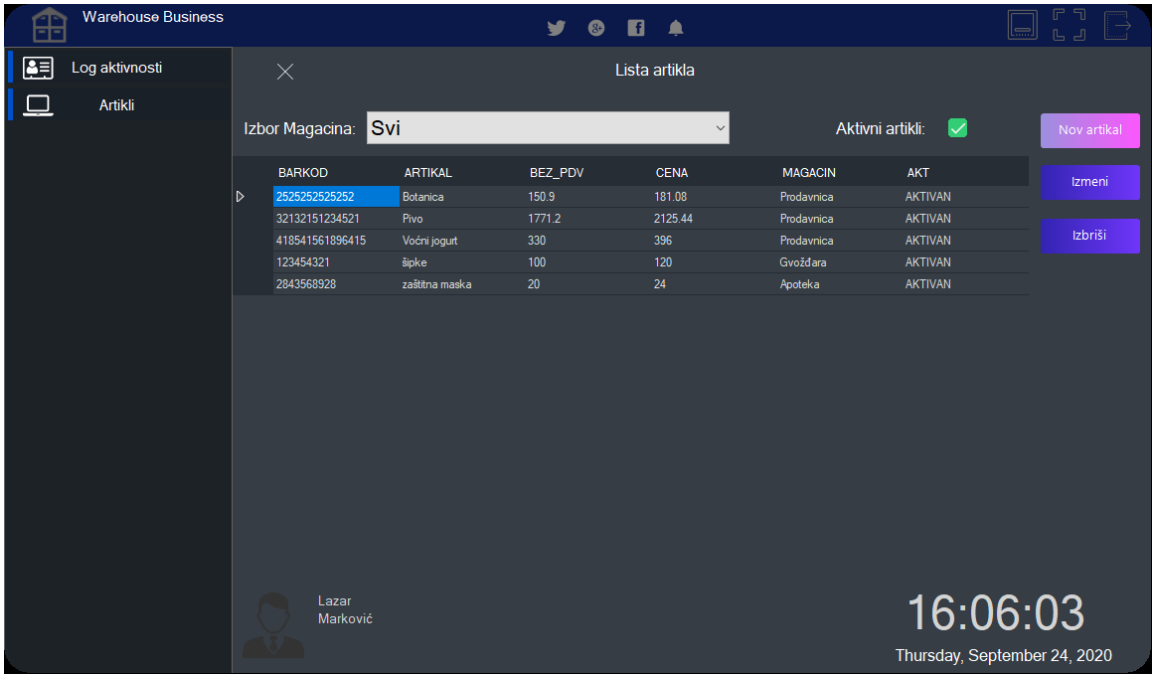


Слика 11. Унос робе у магацин

```
SET SQL DIALECT 3;
/******************************************************************************/
/*                                  Tables                                    */
/******************************************************************************/
CREATE GENERATOR GEN_UNIT_OF_MEASUREMENT_ID;
CREATE TABLE UNIT_OF_MEASUREMENT (
    ID        INTEGER NOT NULL,
    NAME      VARCHAR(100) CHARACTER SET UTF8,
    ALIAS     VARCHAR(100) CHARACTER SET UTF8,
    "ACTIVE"  SMALLINT DEFAULT 0 NOT NULL
);
/******************************************************************************/
/*                                Primary keys                                */
/******************************************************************************/
ALTER TABLE UNIT_OF_MEASUREMENT ADD CONSTRAINT PK_UNIT_OF_MEASUREMENT PRIMARY KEY (ID);
/******************************************************************************/
/*                                  Triggers                                  */
/******************************************************************************/
SET TERM ^ ;
/******************************************************************************/
/*                            Triggers for tables                             */
/******************************************************************************/
/* Trigger: UNIT_OF_MEASUREMENT_BI0 */
CREATE OR ALTER TRIGGER UNIT_OF_MEASUREMENT_BI0 FOR UNIT_OF_MEASUREMENT
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
 IF (NEW.ID IS NULL) THEN NEW.ID = GEN_ID(GEN_UNIT_OF_MEASUREMENT_ID, 1);
END
^
SET TERM ; ^
/******************************************************************************/
/*                            Fields descriptions                             */
/******************************************************************************/
COMMENT ON COLUMN UNIT_OF_MEASUREMENT."ACTIVE" IS
'ako je 0 onda je aktivan ako je 1 onda nije';
```

Слика 12. SQL код за креирање табеле unit_of_measurement

Ако корисник одабере опцију *“Artikli” >>>“Novi artikal”* добија изглед као на слици 13.

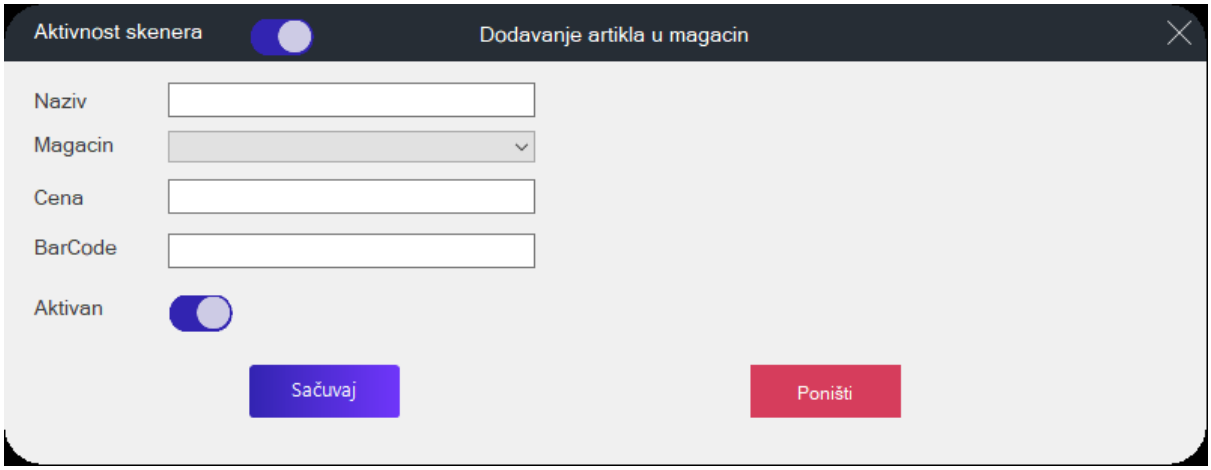


Слика 13. Листа свих артикла у магацину

```
SET SQL DIALECT 3;
CREATE GENERATOR GEN_ARTICLE_ID;
CREATE TABLE ARTICLE (
    ID          INTEGER NOT NULL,
    BARCODE     VARCHAR(255) CHARACTER SET UTF8 NOT NULL,
    NAME        VARCHAR(50) CHARACTER SET UTF8 NOT NULL,
    PRICE       DOUBLE PRECISION NOT NULL,
    MARKET_ID   INTEGER NOT NULL,
    "ACTIVE"    INTEGER DEFAULT 0 NOT NULL
);
ALTER TABLE ARTICLE ADD CONSTRAINT UNQ1_ARTICLE UNIQUE (ID);
ALTER TABLE ARTICLE ADD CONSTRAINT PK_ARTICLE PRIMARY KEY (ID, MARKET_ID);

ALTER TABLE ARTICLE ADD CONSTRAINT FK_ARTICLE_2 FOREIGN KEY (MARKET_ID) REFERENCES MARKET_PLACE (ID);

SET TERM ^ ;

CREATE OR ALTER TRIGGER ARTICLE_BI0 FOR ARTICLE
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
 IF (NEW.ID IS NULL) THEN NEW.ID = GEN_ID(GEN_ARTICLE_ID, 1);
END
^
SET TERM ; ^

COMMENT ON COLUMN ARTICLE."ACTIVE" IS
'ako je 0 znaci da je aktivan ako 1 znaci da nije';
```

Слика 14. SQL код за креирање табеле article

Поред избора магацина постоји могућност чекирања да се погледају активни и неактивни артикли. Ова опција је стављена да се не би пунила беспотребно једна база са артиклима који нам тренутно нису потребни. Такође, могуће је додати нов артикал, изменити постојећи или га обрисати. Уколико корисник одабере опцију *"Nov artikal"* отвара се форма као на слици 15.



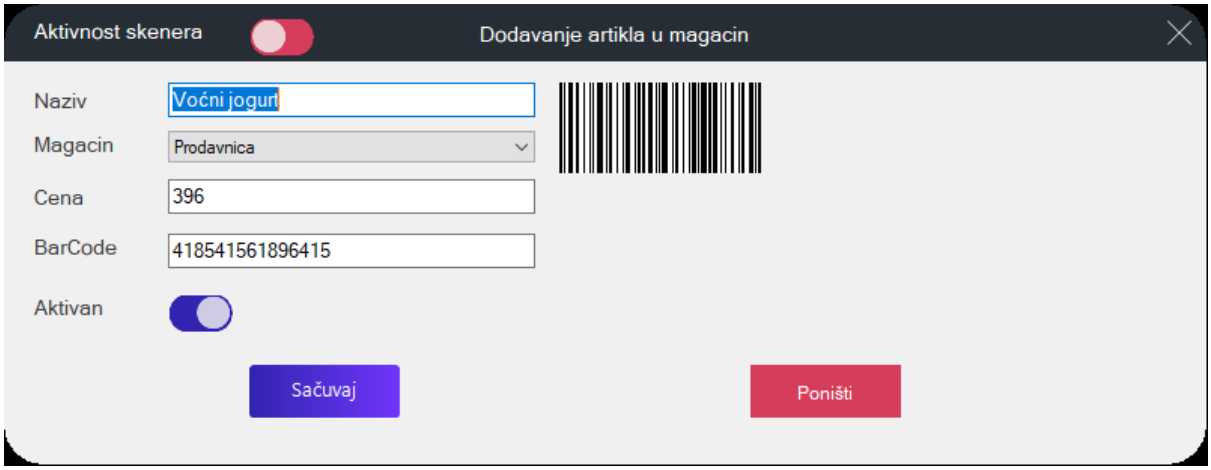
Слика 15. Додавање новог артикла у магацин

Као што је приказано на претходној слици, прва опција *"Aktivnost skenera"* нам аутоматски даје одоговор да ли је скенер активан тј. спреман да очита *"BarCode"* са артикла ког додајемо у базу. Затим се од корисника очекује да додели назив артикла, одабере у ком делу магацина га смести, упише цену, док се у пољу *"BarCode"* аутоматски уписује код који скенер очита али постоји и могућност ручног уношења уколико скенер закаже. Уколико корисник попуни сва поља, кликом на дугме *"Sačuvaj"* артикал ће бити уписан у базу података.

Скенер који је коришћен дат је на слици 16.

Слика 16. Скенер за очитавање бар-кода [7]

Поред опције *"Nov artikal"* имамо могућност да га изменимо или обришемо, потребно је само селектовати који артикал желимо и кликнути на *"Izmeni"* или *"Obriši"*. Ако изаберемо *"Izmeni"* отвара нам се форма са унетим подацима као на слици 17. где корисник мења одговарајуће поље.

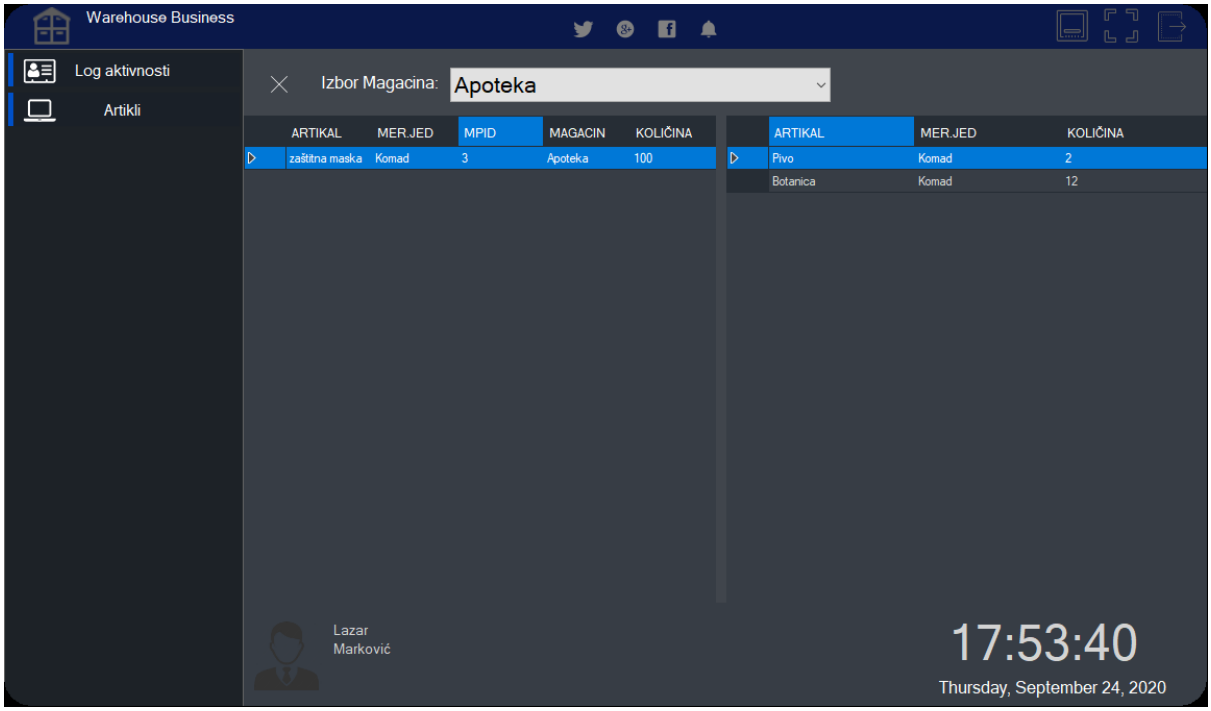


Слика 17. Форма за измену артикла

Ако корисник одабере опцију *"Artikli" >>>"Škart"* која је направљена уколико се неки од артикла оштетио у току превоза или сл. и не може да уђе у продају, бројно стање се смањује у магацину, изглед је дат на слици 19.

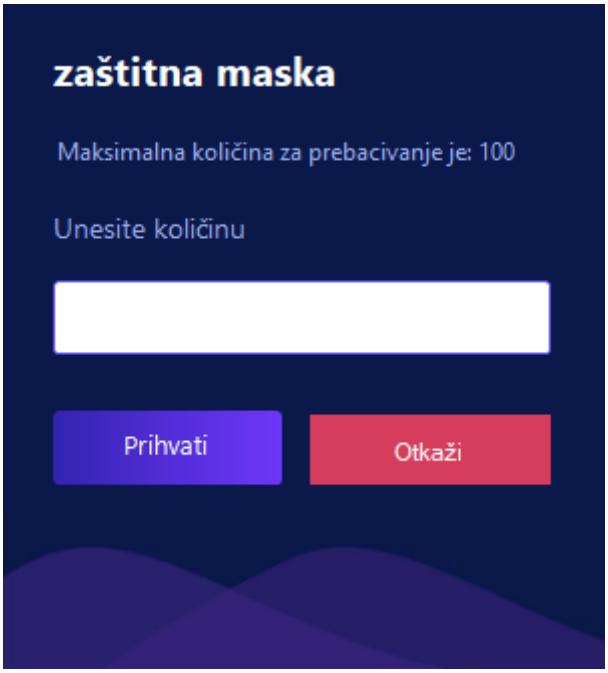
```
SET SQL DIALECT 3;

CREATE GENERATOR GEN_JUNK_ID;

CREATE TABLE JUNK (
    ID          INTEGER NOT NULL,
    ARTICLE_ID  INTEGER NOT NULL,
    UOM_ID      INTEGER NOT NULL,
    QUANTITY    INTEGER
);

ALTER TABLE JUNK ADD CONSTRAINT PK_JUNK PRIMARY KEY (ID, ARTICLE_ID, UOM_ID);

ALTER TABLE JUNK ADD CONSTRAINT FK_JUNK_1 FOREIGN KEY (ARTICLE_ID) REFERENCES ARTICLE (ID);
ALTER TABLE JUNK ADD CONSTRAINT FK_JUNK_2 FOREIGN KEY (UOM_ID) REFERENCES UNIT_OF_MEASUREMENT (ID);

SET TERM ^ ;

CREATE OR ALTER TRIGGER JUNK_BI0 FOR JUNK
ACTIVE BEFORE INSERT POSITION 0
AS
begin
   IF (NEW.ID IS NULL) THEN NEW.ID = GEN_ID(gen_junk_id, 1);
end
^
SET TERM ; ^
```

Слика 18. SQL код за креирање табеле junk



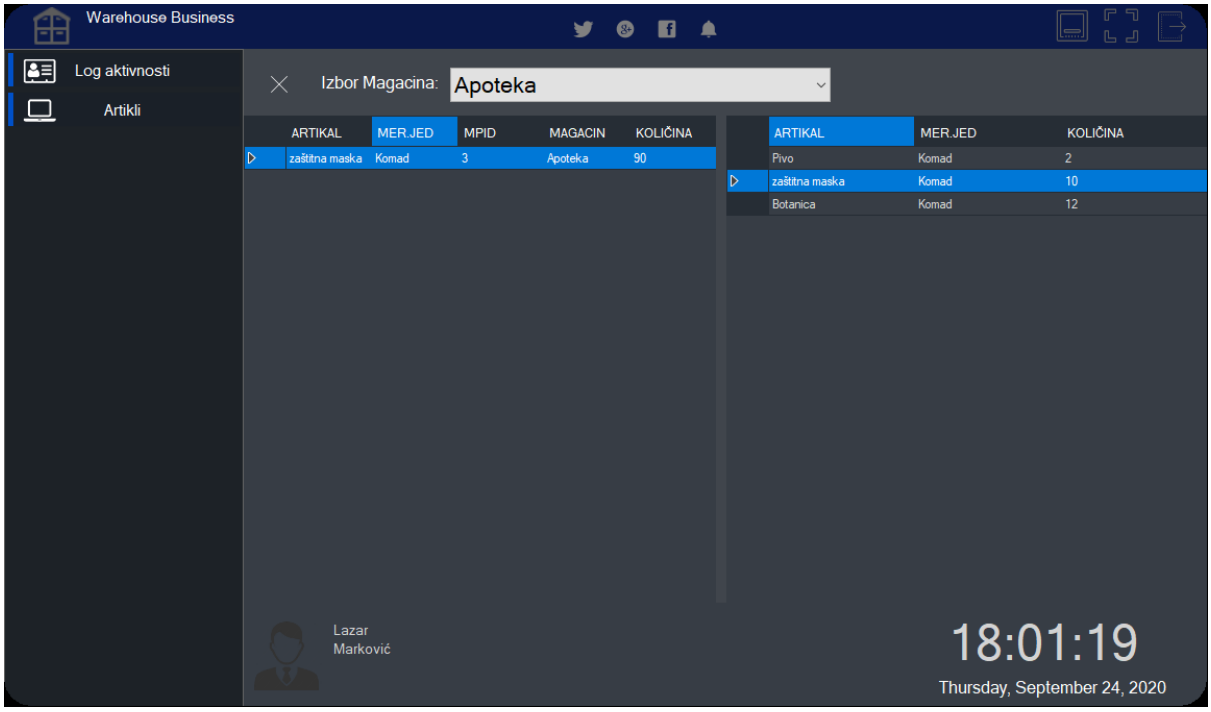
Слика 18. Уношење шкарт артикла

Након избора магацина, потребно је превући селектовани артикал у десну страну ове форме приказа, након чега се појављује форма као на слици 19.



Слика 20. Форма за унос шкарт артикла

Након исправно унете количине од 0 до максималне количине за пребацивање, шкарт артикли се смештају у одговарајућу базу и број у складишту се смањује за унети број у овој форми као на слици 21.



Слика 21. Приказ успешно додатог шкарта

8. Закључак

Након свега наведеног може се закључити и видети директна примена и предности које један информациони систем омогућава. Његове функције и делатности у најбољем светлу виде и користе радници међутим поред радника у оваквим делатностима, овакви системи у многоме олакшавају посао економском делу компаније, људима који се баве наручивањем, дистрибуцијом али и локацијом производа. Долази се до закључка да су овакви системи данас доведени до савршенства уколико оно постоји, јер у само пар кликова долазимо до информација о производима, улазима и излазима из магацина као и уопштено магацинског пословања. Очекује се да ће систем који је развијен уз потребну надоградњу наћи примену у неком објекту и даље помоћи и унапредити делатности истог.

9.Литература

[1] Лазаревић Б.: Базе података, ФОН Београд, Београд 2003.

[2] Павловић-Лажетић Г.: Основе релационих база података, Математички факултет, Београд, 2000.

[3] R. Elmasri, S. Navathe, Fundamentals of Database Systems, Addison-Wesley, Boston, 2003.

[4] Преузето: http://www.ibexpert.net/ibe/uploads/Doc/all.html датум приступа 22.09.2020.

[5] Преузето: https://firebirdsql.org/file/documentation/pdf/en/refdocs/fblangref25/firebird-25-language-reference.pdf датум приступа 22.09.2020.

[6] Преузето: https://vit-vladimir.ru/bs/visual-studio-community-bez-registracii-opisanie-programmnogo-obespecheniya/ датум приступа 22.09.2020.

[7] Преузето: https://www.honeywellaidc.com/products/barcode-scanners/healthcare/enhanced-xenon-1900h-1902h датум приступа 22.09.2020.