Utjecaj odabira baze podataka na brzinu rada programskih aplikacija

Mihalić, Petar

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:812096

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Petar Mihalić

UTJECAJ ODABIRA BAZE PODATAKA NA BRZINU RADA PROGRAMSKIH APLIKACIJA

DIPLOMSKI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Petar Mihalić

Matični broj: 0016136283

Studij: Informacijsko i programsko inženjerstvo

UTJECAJ ODABIRA BAZE PODATAKA NA BRZINU RADA PROGRAMSKIH APLIKACIJA

DIPLOMSKI RAD

Mentor/Mentorica:

Izv. prof. dr. sc. Mario Konecki

Varaždin, kolovoz 2023.

Petar Mihalić

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

U ovom radu je obrađena tema utjecaja odabira baze podataka na brzinu rada programskih aplikacija. Nakon uvoda u temu opisane su metode i tehnike rada kod kojih je opisano da će aplikacija biti izrađena u Visual Studio-u korištenjem ASP.NET Core okvira i C# programskog jezika. Prikazane se i opisane sve funkcionalnosti aplikacije kako bi se znalo na čemu se uspoređuju baze podataka. Kreirana aplikacija je online trgovina nazvana „SneakerShop“. Prikazan je i opisan model podataka koji se koristi kod kreiranja same aplikacije i kod implementacija baza podataka. Opisane su korištene baze podataka, SQLServer, Cassandra, MongoDB, MySQL i PostgreSQL. Za svaku aplikaciju je opisana instalacija potrebnih programa i alata, kreiranje same baze podataka, povezivanje baze s online trgovinom te su prikazane i opisane naredbe za čitanje, kreiranje, ažuriranje i brisanje podataka iz baze. Nakon upoznavanja s aplikacijom i bazama podataka opisan je način mjerenja brzine aplikacije, prikazana implementacija mjerenja u kodu te zapis u datoteku, opisani su testni podaci i procedura mjerenja. Na kraju su izmjereni podaci uređeni kako bi se mogli koristiti za analizu. Analizom rezultata su prokomentirani detaljni rezultati, napravljeno je nekoliko grafova i zaključeno je da za odabranu aplikaciju i implementaciju najbrže rezultate ima MongoDB baza podataka.

Ključne riječi: baza podataka; aplikacija; brzina; analiza; SQLServer; MySQL; PostgreSQL; Cassandra; MongoDB

Sadržaj

1. Uvod

Ovaj rad obradit će temu utjecaja odabira baze podataka na brzinu rada programskih aplikacija. Odabrao sam ovu temu jer mi se svidio praktični dio rada budući da uključuje izradu cijele aplikacije pri čemu ću steći dodatno iskustvo i znanje te se sama tema bavi bazama podataka na koje nisam imao veliki fokus na studiju jer sam odabrao informacijsko i programsko inženjerstvo. Ovim radom nastojim upotpuniti svoja znanja kako bi bio bolje spreman za rad u struci te kako bi se bolje upoznao sa trenutno najkorištenijim bazama podataka. Također sloboda u izboru aplikacije koja se izrađuje mi omogućava odabir programa i okolina za razvoj u kojima najviše znam raditi i u kojima bi htio započeti karijeru.

Tema rada je razrađena paralelno mojem pristupu istraživanju i izradi praktičnog dijela, te sukladno time rad započinje opisom programa i tehnologija koje će biti korištene. Što se tiče razvojnog okruženja želio sam raditi u VisualStudio-u pa je istraživanje o tehnologijama prirodno krenulo u tom smjeru. Odabrao sam ASP.NET Core kao okvir za izradu aplikacije jer smatram da je vrlo popularan danas i korisno je imati iskustvo u tom okviru a nismo ga koristili na studiju pa sam ga spreman naučiti. Nakon opisa tehnika rada i alata korištenih za kreiranje i upravljanje bazama podataka opisat ću programsku aplikaciju koja će biti online trgovina za tenisice naziva „SneakerShop“. Bit će opisane glavne funkcionalnosti aplikacije, prikazani svi dijelovi i pogledi koje aplikacija nudi te će uz to biti prikazan i opisan model podataka koji je zapravo temelj za implementaciju baza podataka. Odabrao sam implementirati i testirati utjecaj sljedećih baza podataka: SQLServer, MongoDB, Cassandra, MySQL i PostgreSQL. Pobliže ću objasniti svaku od baza podataka te prikazati njihovu instalaciju, kreiranje i povezivanje sa online trgovinom. Prikazat ću i CRUD operacije kako bi se vidjele razlike i sličnosti kod sintakse upita. Što se tiče samog mjerenja brzine aplikacije prikazat ću kako sam implementirao mjerenje i zapisivanje izmjerenih podataka u datoteku, opisat ću koje i koliko testnih podataka sam koristio, te objasniti proceduru testiranja koju sam slijedio kako bi uvjeti za svaku bazu bili što više standardizirani. Prikupljeni podaci će biti obrađeni za lakšu analizu te ću prikazati rezultate za svaku pojedinačnu bazu podataka, napraviti zanimljive usporedbe te prikazati i zaključiti finalne rezultate ovog rada.

Ovim radom nastojim steći uvid u implementaciju i upravljanje različitim bazama podataka te steći dodatno iskustvo u pisanju C# koda korištenjem Visual Studio-a. Neka od glavnih pitanja na koje se nadam pronaći odgovor ovim radom su naravno koliki je zapravo utjecaj baze podataka na brzinu i jesu li razlike u brzini dovoljno značajne da se isplati učiti raditi s drugačijom logikom, mijenjati model podataka i žrtvovati lakoću upravljanja bazom i jednostavnost integracije za dodatnu brzinu.

2. Metode i tehnike rada

Prvi korak kod razrade ove teme je bilo donošenje odluke o vrsti programske aplikacije koju ću napraviti za testiranje baza podataka. Odlučio sam se na online trgovinu tenisicama jer sam smatrao da takva aplikacija sadrži funkcionalnosti i podatke adekvatne za testiranje baza podataka te smatram da mogu sa osnovnim istraživanjem kreirati realističan model podataka. S obzirom da tema nudi slobodu u odabiru alata, programskih rješenja, programskog jezika i vrsti aplikacije, odlučio sam se koristi Visual Studio-om. Istraživanjem o izradi web aplikacije u Visual Studio-u otkrio sam ASP.NET Core okvir za razvoj idealan za potrebe ovog rada i bio sam spreman naučiti raditi u njemu. Slijedeći službene upute za kreiranje prve web aplikacije u ASP.NET Core-u upoznao sam se i sa SQL Serverom koji je odlično integriran u Visual Studio-u i to je ujedno postala prva baza podataka ovog rada i sa njom sam uradio par testnih programa da se upoznam sa procesom i usput sam istražio mogućnost povezivanja brojnih drugih baza podataka s ASP.NET Core projektom. U nastavku ću ukratko objasniti navedeni alat i okvir te razne alate i načine za kreiranje i upravljanje bazama podataka koje sam odabrao.

2.1. Visual Studio

Visual Studio je integrirano razvojno okruženje (eng. *Integrated development evnironment - IDE*) koje je odlično za pisanje i uređivanje koda, pronalazak grešaka te objavljivanje aplikacija. Ovo razvojno okruženje uključuje kompajlere, grafičke dizajnere, IntelliSense za inteligentno dovršavanje koda, pomoć pri kodiranju te prikaz raznih informacijama o objektima i parametrima te brojne druge korisne značajke koje olakšavaju proces izrade aplikacija [1][2].

Odlučio sam odabrati ovaj alat za kreiranje web aplikacije jer sam dobro upoznat sa radom u programu i mislim da je najbolje okruženje za pisanje koda u C# jeziku koji je također moji primarni jezik za programiranje. Uz navedene prednosti Visual Studio nudi i brojna proširenja i nadogradnje na osnovni program ovisno o potrebama korisnika i vrsti aplikacije. Za potrebe ovog projekta koristit ću Visual Studio Community 2022 uz kojeg sam odabrao instalirati modifikaciju „*ASP.NET and web development*“ koja uključuje razvojne alate za .NET okvir (eng. framework) i ASP.NET i web razvojne preduvjete poput Entity Frameworka i SQL Server lokalne baze. Uz navedenu modifikaciju Visual Studio nudi i instalaciju NuGet paketa, dijelova koda kreiranih od strane drugih programera koji su dostupni za korištenje u projektu i puno olakšavaju određene implementacije te su gotovo neophodni za integriranje različitih baza podataka kao u ovom radu.

2.2. ASP.NET Core

ASP.NET Core je višeplatformski okvir za razvoj modernih aplikacija povezanih na Internet i oblak, otvorenog je koda i ima odlične performanse. Okvir omogućuje razvoj web aplikacija i servisa, aplikacije za Internet stvari (eng. *Internet of Things - IoT*) i mobilni backend. Rad u ASP.NET-u se primarno dijeli na tri vrste pristupa programiranju aplikacije, a to su korištenje MVC uzorka dizajna, Razor stranica (eng. *Razor Pages*) ili Blazor okvira. [3]

Glavna razlika ova tri pristupa je mjesto renderiranja korisničkog sučelja. MVC i Razor stranice renderiraju sučelje s poslužitelja, servera, dok Razor služi za kreiranje interaktivnog web sučelja na strani klijenta. Postoji i mogućnost korištenja kombiniranog pristupa što ima svoje određene prednosti no u ovom ću se radu koristi Razor stranicama. [4] Testirao sam razlike u korištenju MVC-a i Razor stranica i odabrao sam Razor stranice zbog boljeg pregleda koda, manje klasa i manje krivulje učenja za početnike u ovom okviru.

Nadodao bih da sam zajedno s ASP.NET Core-om i Razor stranicama koristio i Entity Framework Core te da su svi osnovni dijelovi logike programiranja u ovom okviru odlično objašnjeni i potkrijepljeni uputama i primjerima u službenoj Microsoft dokumentaciji.

2.3. Upravljanje bazama podataka

Za kreiranje i upravljanje bazama podataka svaka vrsta ima razlike i ovih 5 baza definitivno ne slijede istu shemu. Cilj ovog poglavlja je ukratko objasniti alate koji se koriste kod ovih baza podataka. Naglasio bih da mi je cilj kod svake instalacije bio odabrati najjednostavniju lokalnu verziju baze kako dodatni programi za upravljanje i spajanja na druge servere ne bi imalo utjecaj na konačne rezultate ovog rada.

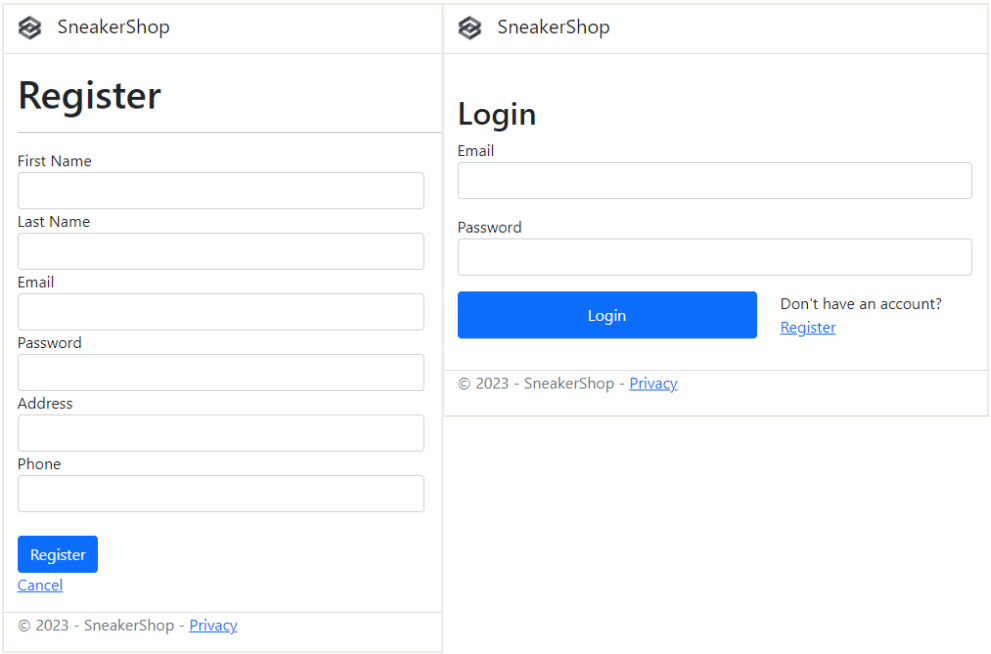
SQL Server dolazi već instaliran sa ASP.NET Core modifikacijom za Visual Studio kao što sam već spomenuo, točnije koristi se LocalDB koji predstavlja laganiju verziju SQL Server Express Database Engine-a [5]. Tako da se kreiranje ove baze i njezino upravljanje vodi isključivo kroz Visual Studio i nisu potrebne dodatne instalacije i alati. Za MongoDb i Cassandru je preuzeta datoteka sa interneta, te su dodane sistemske varijable za pokretanje baza podataka i njihovih ljuski (eng. *shell*) preko naredbenog retka (eng. *Command Prompt*). Za instalaciju i pokretanje MySQL baze korišten je XAMPP program te alat phpMyAdmin koji služi za administriranje MySQL baze preko web preglednika. Za instalaciju PostgreDB-a koristio sam EnterpriseDB instalaciju koja uz lokalni server omoguće i instalaciju pgAdmin 4 alata za grafičko upravljanje bazom podataka. Za sve baze su instalirani prikladni NuGet paketi u Visual Studio-u koji omogućuju spajanje sa aplikacijom i kreiranje upita na bazu.

3. Aplikacija

Programska aplikacija u ovom radu predstavlja temeljni stup za testiranje i ostatak rada će ovisiti o tipu aplikacije i modelu podataka koji će aplikacija koristiti. Jedno od prvih istraživanja i koraka pri razradi ovog rada bilo je određivanje na kakvoj će se aplikaciji bazirati usporedba baza podataka. Odlučio sam se za online trgovinu nazvanu „SneakerShop“ jer smatram da je online trgovina vrsta aplikacije koju je gotovo svatko koristio u današnje doba i takve aplikacije već imaju pred određene i očekivane funkcionalnosti. U ovom poglavlju ću prikazati sve funkcionalnosti koje aplikacija ima, potkrijepit ću sve dijelove aplikacije slikama ekrana te opisati logiku iza implementiranih elemenata. Nakon razumijevanja rada aplikacije pobliže ću objasniti model podataka koji je korišten i zašto je tako oblikovan. Na kraju poglavlja ću prikazati implementaciju jedne od stranica u Visual Studio-u kako bi dobili cjelokupnu sliku o aplikaciji prije implementacija različitih baza podataka.

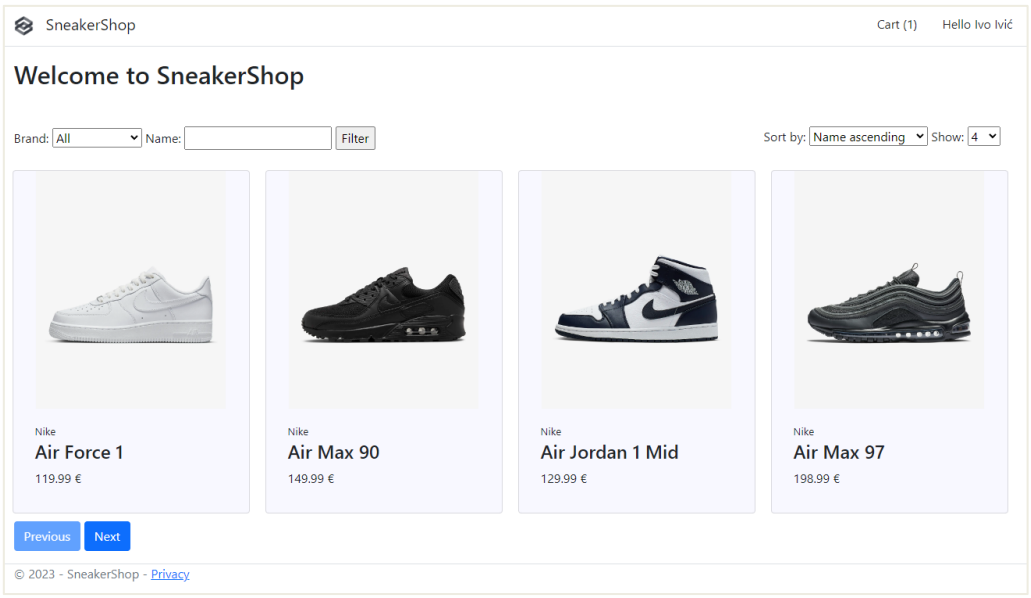
3.1. Funkcionalnosti

Aplikaciju je moguće koristiti kao registrirani i prijavljeni korisnik ili samo kao gost. Naravno kao gost ograničena je akcija poput posjeta stranice profila i pregleda povijesti narudžbi ali košarica i ostale radnje potrebne za provođenje kupnje su omogućene. Stoga ću prvo prikazati funkcionalnosti registracije i prijave registriranog korisnika, vidljive na slici 1.



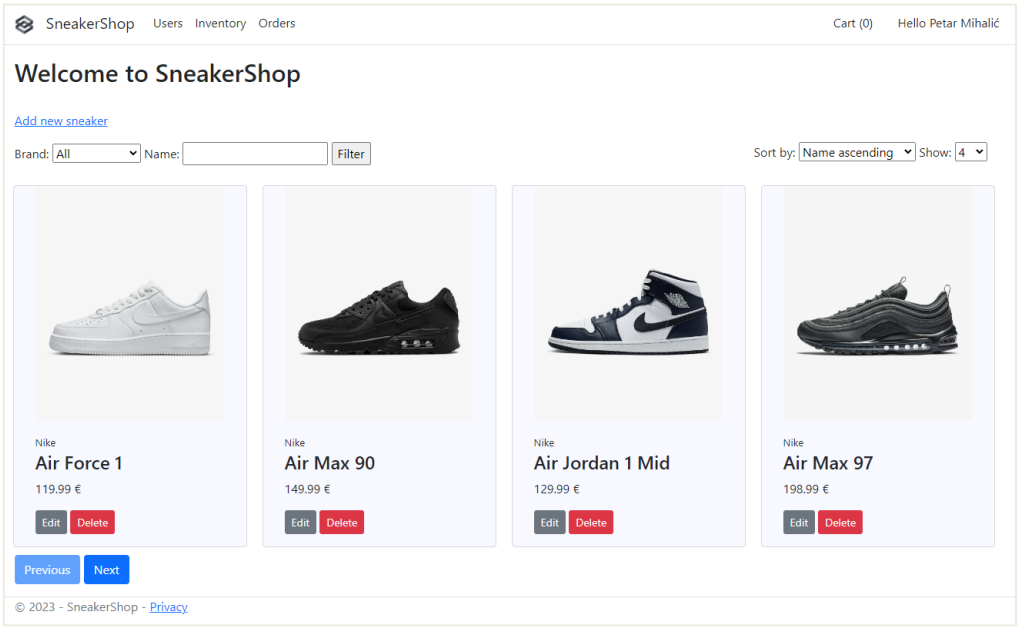
Slika 1: Obrezani izgred stranice za registraciju i stranice za prijavu (vlastita izrada)

Nakon uspješne registracije i prijave, korisniku se prikazuju funkcionalnosti ovisno o njegovoj ulozi. Na sljedećoj slici prikazana je početna stranica aplikacije iz perspektive klasičnog korisnika aplikacije.



Slika 2: Početna stranica, pogled prijavljenog korisnika (vlastita izrada)

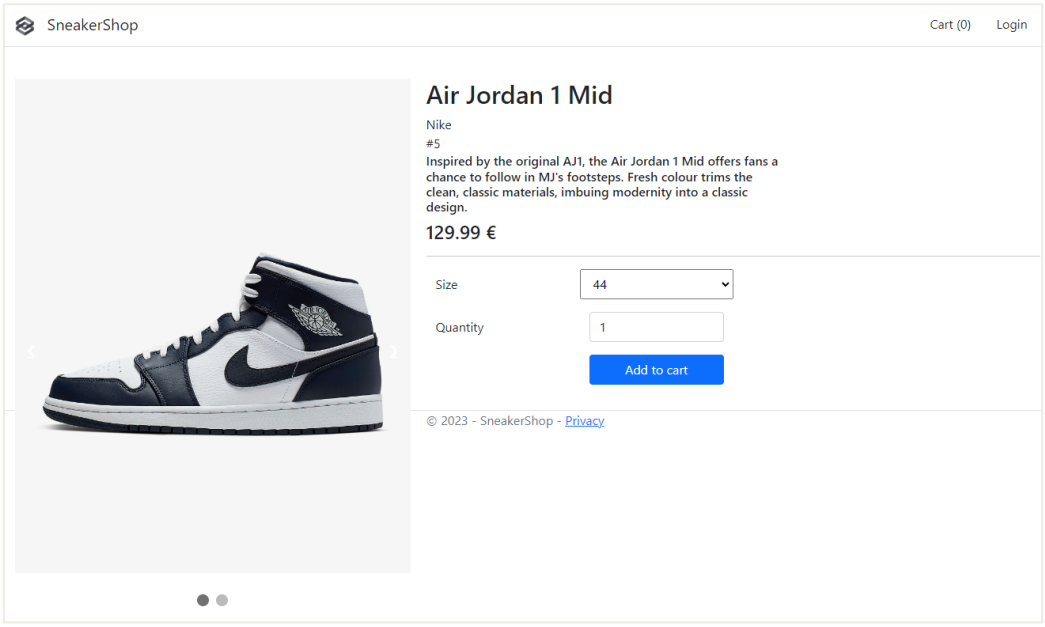
Pogled je gotovo identičan onome prije prijave, jedina je razlika što se umjesto pozdrava u gornjem desnom kutu koji vodi na profil korisnika, nalazi gumb za prijavu. Sljedeća slika prikazuje početnu stranicu ako je prijavljen administrator.



Slika 3: Početna stranica, pogled prijavljenog administratora (vlastita izrada)

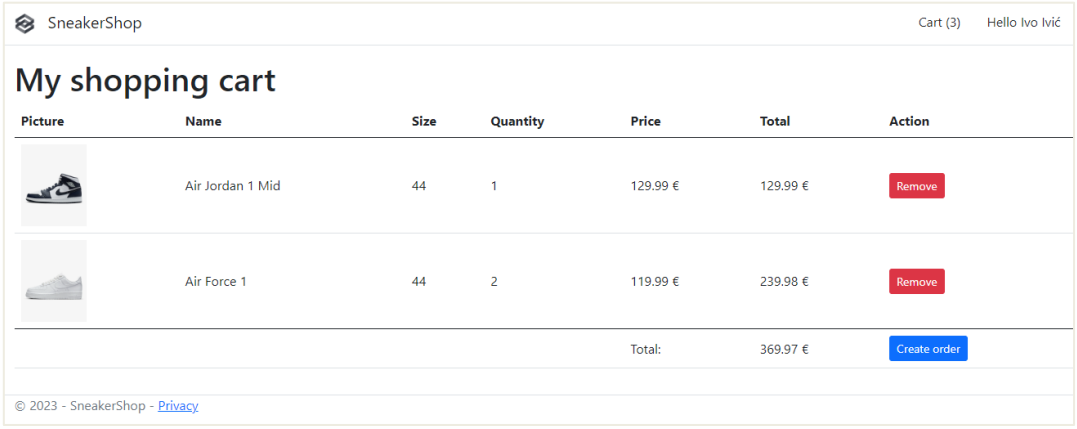
Kada je prijavljen administrator vidljiv je meni za posjet stranica korisnika, inventara i narudžbi (*Users, Inventory, Orders*) te su vidljive poveznice za dodavanje, uređivanje i brisanje proizvoda. Ono što vide administrator, korisnici i gosti su elementi za filtriranje, sortiranje i

odabir količine prikaza proizvoda na stranici. Klikom na bilo koju tenisicu otvara se stranica s detaljima o tenisici prikazana na sljedećoj slici.



Slika 4: Stranica tenisice, pogled gosta (vlastita izrada)

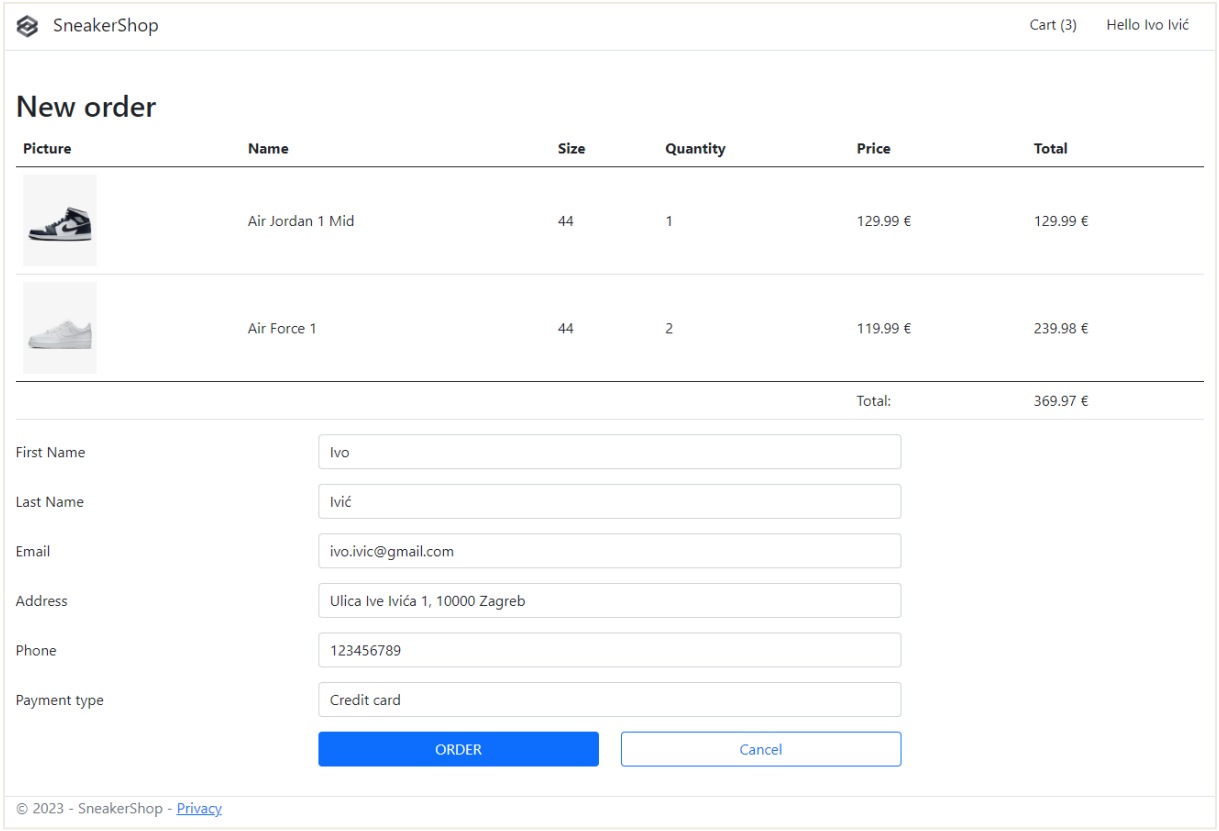
Na stranici tenisice moguć je odabir veličine, u padajućem izborniku nalaze se veličine čija je količina u inventaru veća od nule. A ukoliko korisnik odabere količinu koja je veća od dostupne prikazuje se poruka koja korisnika obavještava o dostupnoj količini za odabranu veličinu. Također se može listati po slikama, zbog jednostavnosti aplikacije i modela svaka tenisica ima točno dvije slike. Nakon dodavanja željene tenisice u košaricu, na gornjoj traci se ažurira broj proizvoda u košarici. Klikom na košaricu otvara se sljedeća stranica:



Slika 5: Košarica sa proizvodima (vlastita izrada)

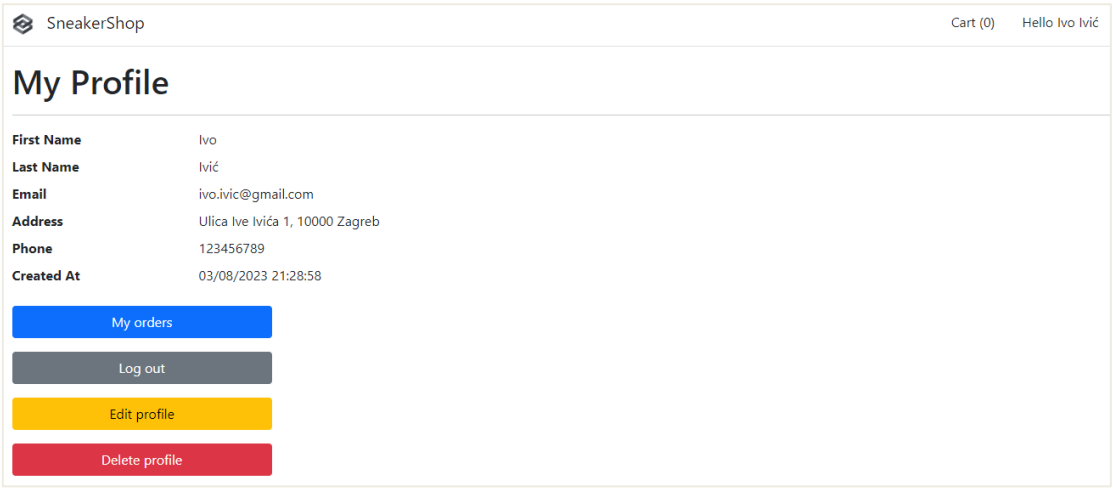
U košarici se vidi suma za pojedine proizvode te na dnu finalna suma cijele košarice. Klikom na gumb *Create order* otvara se nova stranica koja prikazuje podatke iz košarice te sadrži formu za ispunjavanje podataka potrebnih za kreiranje narudžbe. Ako gost dođe na stranicu za naručivanje tada će morati ispuniti formu s osobnim podacima i odabrati oblik

plaćanja, a ako je korisnik prijavljen onda se forma automatski ispunjava podacima sa profila, korisnik naravno može izmijeniti svaki od podataka ako je potrebno i promijeniti oblik plaćanja odabirom jednog od ponuđenih u padajućem izborniku.



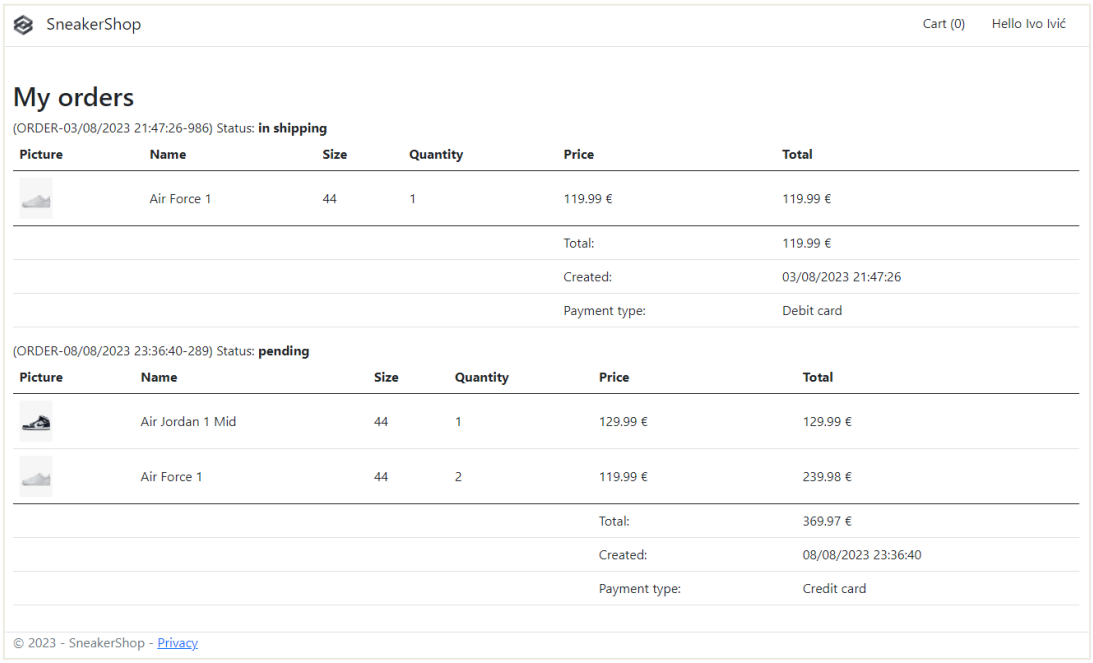
Slika 6: Stranica za kreiranje narudžbe, pogled prijavljenog korisnika (vlastita izrada)

Kreiranjem narudžbe prazni se kartica, smanjuje se inventar za naručenu količinu i narudžba je vidljiva ako korisnik klikne na svoje ime u gornjem desnom kutu i zatim na gumb „*My orders*“. Na sljedećoj slici je prikazana stranica profila na kojoj se uz podatke korisnika nalaze gumbi za pregled narudžbi, odjavu, uređivanje profila i brisanje profila.



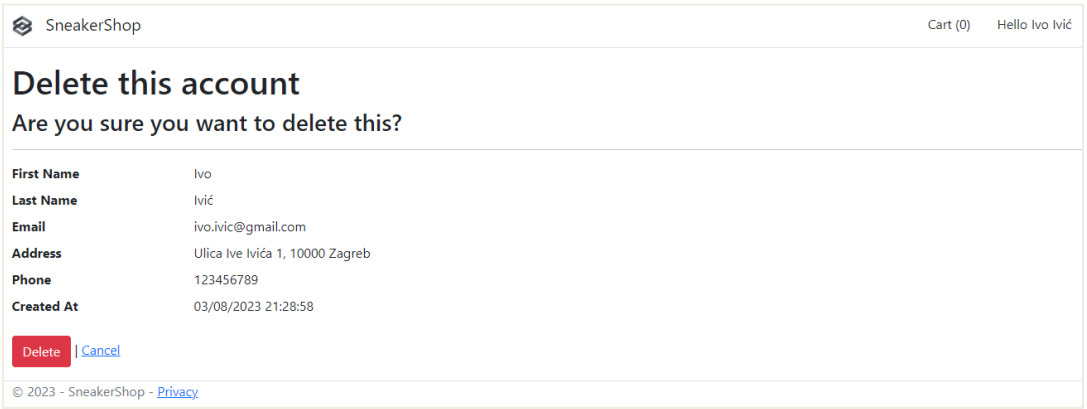
Slika 7: Stranica profila (vlastita izrada)

Klikom na odjavu (eng. *Log out*) briše se trenutna sesija i vraća se korisnika na početnu stranicu gdje je vidljiva poveznica za prijavu. Uređivanje slijedi istu shemu za svaki objekt tako da će ta funkcionalnost biti prikazana za objekt tenisice. Gumb za narudžbe (*My orders*) otvara stranicu koja je prikazana na sljedećoj slici, za svaku narudžbu su prikazani podaci kao što su bili u košarici i kod kreiranja narudžbe, uz to je vidljiv naziv narudžbe koji slijedi određeni format, prikazan je trenutni status narudžbe i vrijeme kada je kreirana.



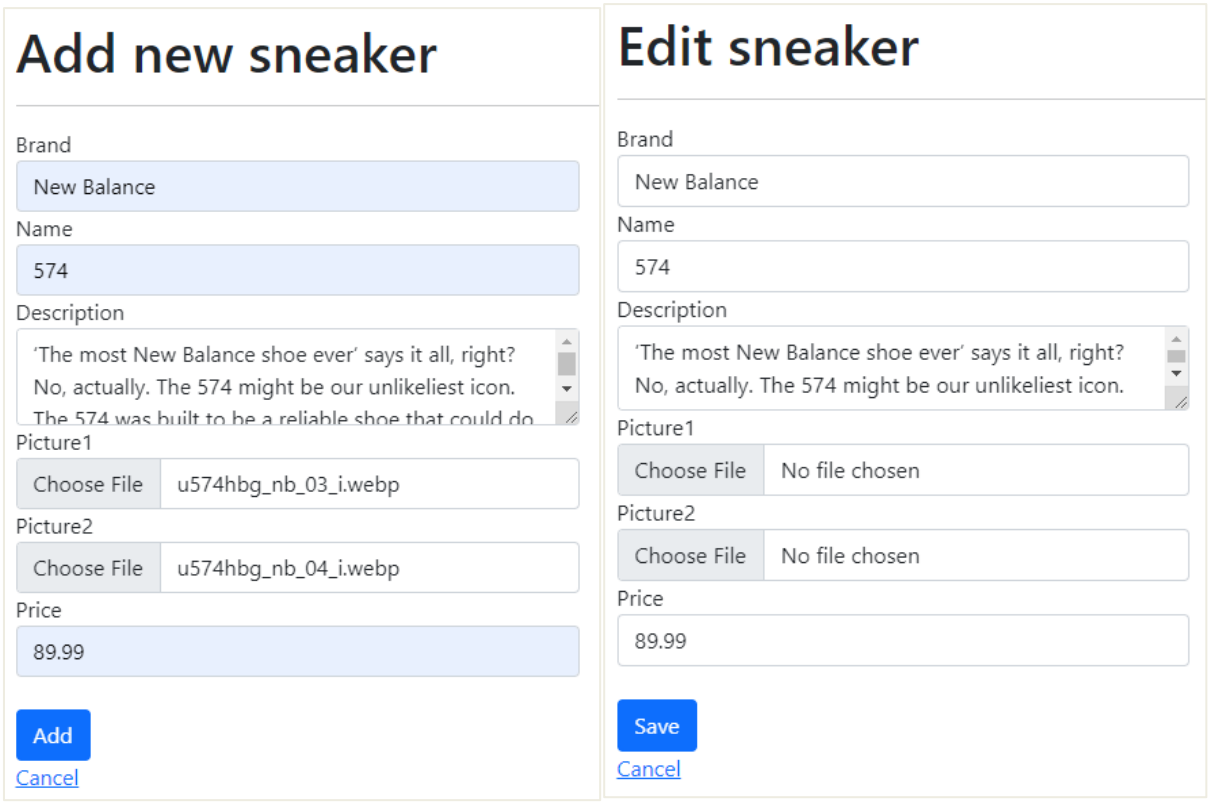
Slika 8: Stranica povijest narudžbi (vlastita izrada)

Gumb za brisanje profila vodi na stranicu za potvrdu željene akcije vidljivu na slici 9. Brisanjem profila korisnik se naravno automatski odjavljuje te se brišu svi njegovi podaci iz baze podataka. To su sve funkcionalnosti dostupne klasičnom korisniku aplikacije.



Slika 9: Stranica za brisanje računa (vlastita izrada)

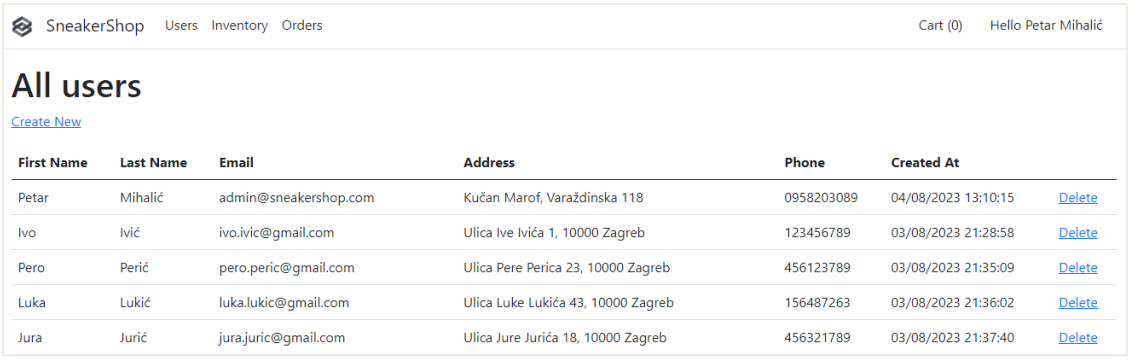
Za nastavak opisa funkcionalnosti bit će prijavljen administrator čiji je pogled početne stranice vidljiv na slici 3. Iz početne stranice administrator može upravljati tenisicama, klikom na poveznicu za dodavanje nove tenisice u ponudu ili uređivanjem postojeće prikazuju se stranice vidljive na sljedećoj slici.



Slika 10: Obrezani izgled stranice za dodavanje i uređivanje tenisice (vlastita izrada)

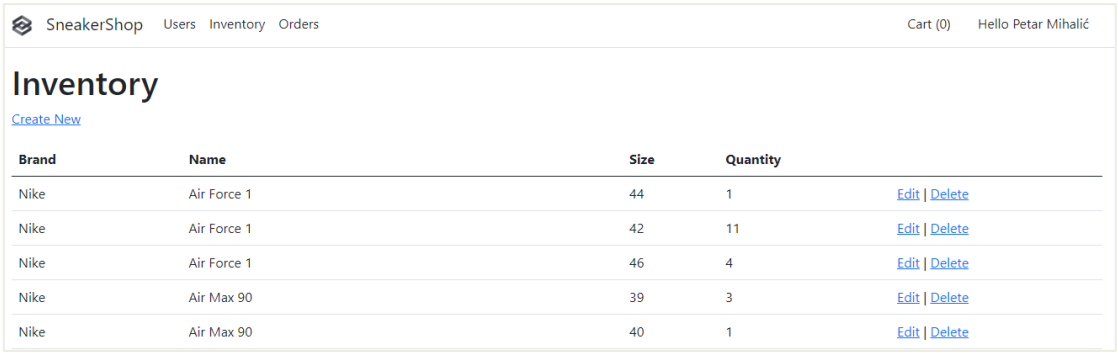
Moguće je prenijeti do dvije slike sa računala, prihvaćaju se svi formati slika. Slike se pri dodavanju nove tenisice spremaju u bazu podataka. Obavezno se mora prenijeti jedna slika dok je polje za drugu sliku jedino koje nije obavezno za ispunjavanje kod unosa tenisice. Kod uređivanja postojeće tenisice slike nisu automatski prikazane no mogu se mijenjati ako se prenesu nove, ukoliko polja ostanu prazna stare slike se neće promijeniti.

Posljednje tri funkcionalnosti koje ima samo administrator se nalaze u gornjoj traci aplikacije a to su stranice korisnici (eng. *Users*), inventar (eng. *Inventar*) te narudžbe (eng. *Orders*). Na stranici svih korisnika, vidljivoj na slici 11, izlistani su svi podaci za korisnike osim lozinke, ova stranica služi za provjeru i ukoliko je došlo do problema administrator može sam obrisati bilo kojeg korisnika.



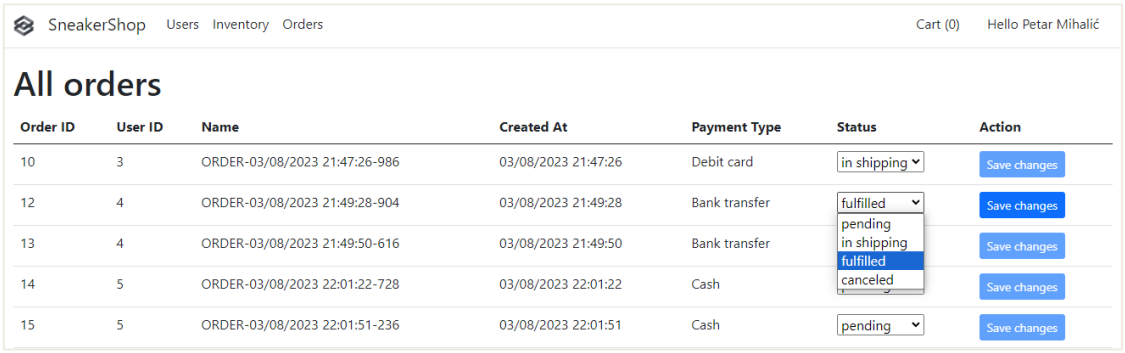
Slika 11: Stranica svih korisnika, prikazan dio popisa (vlastita izrada)

Sljedeća stranica je inventar online trgovine, ovdje je prikazano koliko je proizvoda određene veličine dostupno na skladištu. Administrator može dodavati novu veličinu i dostupnu količinu te veličine za postojeći proizvod, uz to može povećati ili smanjiti količinu već postojeće veličine tenisice ili kompletno obrisati veličinu iz baze podataka.



Slika 12: Stranica inventara, prikazan dio popisa (vlastita izrada)

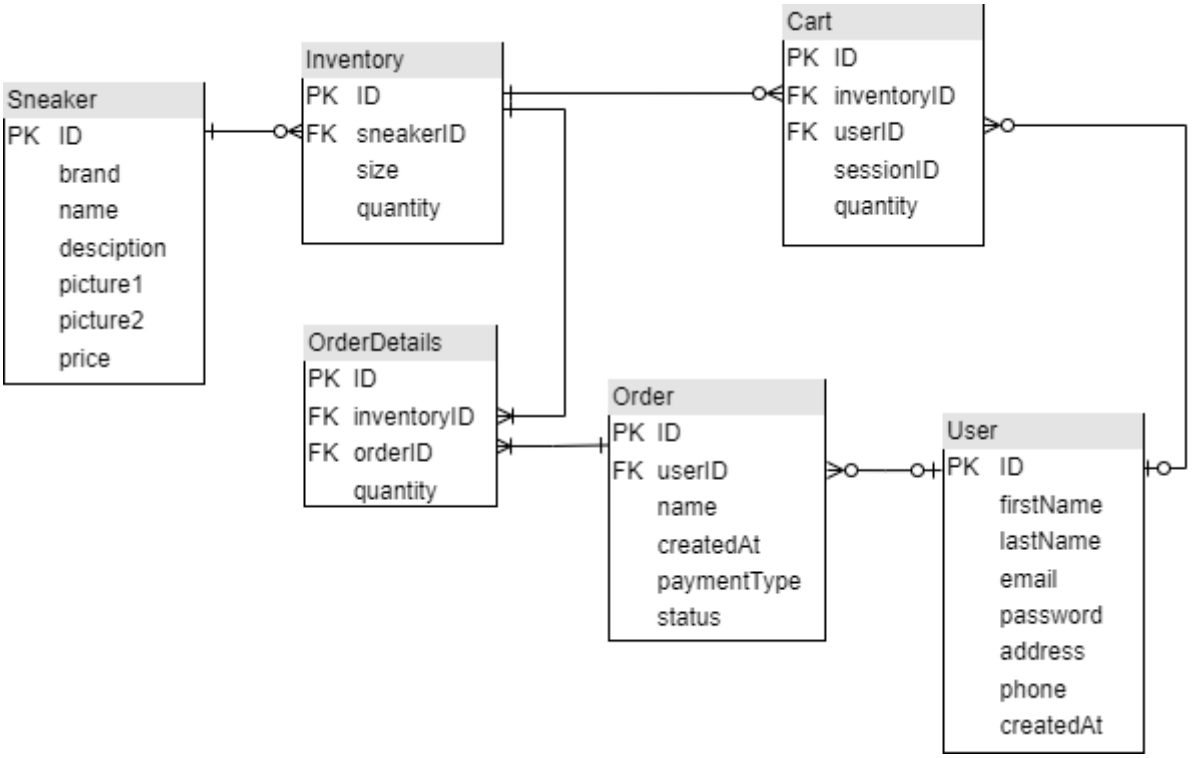
Posljednja funkcionalnost administratora je pregled i upravljanje narudžbama, na stranici vidljivoj na slici 13 izlistane su sve narudžbe, prikazan je njihov ID, ID korisnika ili „gost“ ukoliko korisnik nije bio prijavljen, naziv narudžbe, datum kreiranja, oblik plaćanja i status. Moguće je mijenjati status narudžbe sukladno njezinom ispunjavanju što onda vidi i može pratiti korisnik kao na slici 8.



Slika 13: Stranica svih narudžbi, prikazan dio popisa (vlastita izrada)

3.2. Model podataka

Nakon određivanja vrste aplikacija glavno je pitanje postalo koje podatke koristiti te kako ih oblikovati i povezati. Smatram da je model podataka temeljni stup ovog rada jer se oko njega gradi cijela aplikacija te se modeliraju baze podataka kako bi mogle pohraniti i raditi sa odabranim podacima na svoj način. Budući da sam prvo izradio aplikaciju koristeći SQL Server bazu podataka, podatke sam također oblikovao na način prihvatljiv za to okruženje. Zajedno sa testiranjem različitih načina izrade aplikacije, testirao sam i različite modele podataka dok nisam završio s modelom koji je spremao podatke na način kojeg sam smatrao zadovoljavajućim i efikasnim za potrebe aplikacije. Kod modeliranja podataka uz samo testiranje i izradu modela iz glave koristio sam i određene izvore s interneta kako bi potvrdio svoje ideje i/ili dobio ideje za poboljšavanje [6][7].



Slika 14: Model podataka SneakerShop aplikacije (vlastita izrada)

Na slici iznad je prikazan model koji sam koristio kao referencu za kreiranje objekata u aplikaciji te za kreiranje različitih baza podataka korištenih u ovom radu. Namjerno na slici nisu napisane vrste podataka jer postoje razlike u implementaciji i vrsti podataka koje baza podataka može primiti. Na primjer u SQL bazama ID je najčešće cijeli broj (eng. *Integer*) dok je kod Cassandre to univerzalno jedinstveni identifikator (eng. *Universal Unique Identifier - UUID*) jer takva baza ne podržava automatsko povećavanje ID polja. Neke baze će imati i

dodatna polja u tablicama s vanjskim ključevima u koja će spremati cijele objekte umjesto da postoji samo veza, više o tome kod povezivanja baza podataka s aplikacijom.

U nastavku ovog poglavlja će biti objašnjena logika iza modela podataka na slici 14. Tablica tenisice (eng. *Sneaker*) sadrži vlastiti ID te polja za brand, naziv tenisice, opis, prvu sliku, drugu sliku i cijenu. Polja za slike su zamišljena kao binarni tip podatka ili najsličnija varijacija koju baza podataka nudi. Polja za veličinu i količinu tenisice su odvojena u zasebnu tablicu inventar (eng. *Inventory*) zato što smatram da nije potrebno za svaku veličinu tenisice zapisivati sve ostale podatke koji se neće mijenjati. Na ovaj način jedna tenisica može imati nula ili više veličina kojih ima određeni broj (količina) dok svaka veličina i količina u inventaru mora obavezno biti vezana za jednu tenisicu.

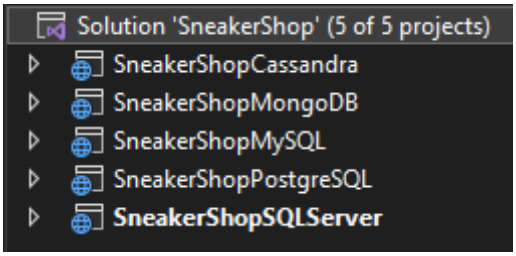
Tablica korisnika (eng. *User*) sadrži osnovne podatke o korisniku, lozinka je kriptirana korištenjem SHA256 algoritma te se u takvom obliku sprema u baze. Korisnik može imati nula ili više narudžbi te nula ili više proizvoda u košarici.

Tablica košarice (eng. *Cart*) sadrži ID košarice, vanjski ključ na tablice inventara i korisnika. Dodano je i polje *SessionID* koje sadrži ID sesije generirane u aplikaciji, ovo polje se popunjava ako gost stavlja tenisice u košaricu jer tada nije dostupan ID korisnika. Košarica uz vanjski ključ na inventar sadrži i vlastito polje za količinu u koja mora biti manja ili ista od količine u inventaru za istu tenisicu. Svaka košarica mora sadržavati barem jedan inventar dok se inventar ne mora nalaziti ni u jednoj košarici.

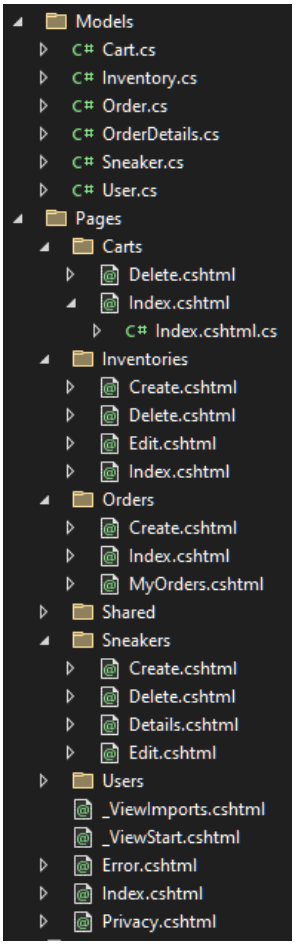
Posljednje dvije tablice su narudžba (eng. *Order*) i detalji narudžbe (eng. *OrderDetails*). Narudžba je slična košarici u smislu da polje s vanjskim ključem prema tablici korisnika može ostati prazno ako narudžbu kreira gost. Uz vlastiti ID narudžba sadrži ime narudžbe koje se generira prema određenoj shemi u aplikaciji kako bi se koristilo za ispis i lakšu identifikaciju narudžbe. Ostala polja narudžbe su datum i vrijeme kreiranja, oblik plaćanja i status. Oblik plaćanja i status imaju pred definirane opcije koje su riješene programski unutar aplikacije tako da nisam smatrao potrebu za dodatnim tablicama koje će sadržavati samo naziv kategorija. Tablica detalja narudžbe je zapravo kopija košarice jer se kod naručivanja košarica prazni a naručene proizvode je i dalje potrebno zapisati. Jedna narudžba mora imati jedan ili više detalj narudžbe, dok redak tablice detalja može biti povezan samo s jednom narudžbom.

3.3. Implementacija

Programska aplikacija ovog rada će sadržavati implementaciju 5 različitih baza podataka. Kako bi kod ostao pregledan, budući da nije potrebno da aplikacija sadrži nekakav oblik mijenjanja između baza podataka tokom rada, svaka će implementacija baze imati svoju aplikaciju. Kako bi osigurao da su sve aplikacije iste po logici, strukturi i svim ostalim aspektima osim baze podataka, prvo je do kraja kreirana aplikacija koristeći SQL Server. Zatim je za svaku sljedeću bazu podataka kopiran cijeli projekt i napravljena potrebna izmjena kako bi aplikacija koristila drugu bazu podataka.



Slika 14: Visual Studio Solution sa 5 verzija iste aplikacije (vlastita izrada)



Na slici 14. je vidljiva struktura rješenja unutar Visual Studio-a. Na taj način je omogućeno lako pokretanje aplikacije s bilo kojom bazom podataka dok je kod pregledan i lako su vidljive razlike između baza podataka.

Svaka aplikacija u rješenju sadrži iste modele i strukturu stranica prikazanu na slici lijevo. Korištenjem Razor Pages pristupa kodiranje scenarija je više orijentirano na stranicu i produktivnije je od korištenja dodatnih klasa za upravljanje i poglede. Dodavanjem Razor stranice automatski se kreiraju dvije datoteke, jedna predstavlja samu stranicu (na slici 15. su to sve datoteke koje završavaju sa .cshtml) dok je druga datoteka klasa modela stranice, ona služi za odvajanje logike stranice od prezentacije, završava sa .cs i strukturno je postavljena kao dijete stranice. U klasi se definira rukovanje zahtjevima poslanim na stranicu i obrađuju se podaci za prikaz na stranici. Najčešće funkcije za rukovanje (eng. *handlers*) su *onGet* i *onPost*. *onGet* se koristi za inicijalizaciju stanja potrebnog za prikaz stranice, ona zapravo prikazuje podatke na stranici, dok se *onPost* koristi za obradu podataka forme nakon podnošenja.

Slika 15. Modeli i struktura stranica aplikacije

4. Baze podataka

Upravljanje i spremanje podataka jedan je od najvažnijih aspekata svakog poslovanja i dobre organizacije. Kako bi podaci bili korisni, lako dostupni i ispravno spremljeni potrebno je koristiti nekakav organizirani pristup grupiranja podataka, te odrediti kakvu vezu podaci imaju jedni s drugima i na koji način bi podaci trebali biti spremljeni kako bi bili od najveće koristi za poslovanje. U prošlosti su se podaci najčešće spremali u fizičkom obliku koji je s vremenom bio sve više standardiziran i počele su se kreirati podatkovne veze između vrsti dokumenata povezanih logikom organizacijskog i poslovnog procesa. Danas postoji mnogo različitih načina za spremanje podataka, što je dovelo do jako puno različitih baza podataka, neke koje se fokusiraju na specifične funkcionalnosti i dominiraju performansama u specifičnim uvjetima i one koje su vrlo fleksibilne i široko korištene te dominiraju tržištem već godinama.

Sam pojam baze podataka je puno korišten u današnje doba te se može definirati na puno načina, jedna od boljih i detaljnijih definicija napisao je Mladen Varga [9]:

„Baza podataka jest kolekcija podataka strukturiranih u skladu s fizičkim te posredno logičkim i konceptualnim modelom podataka informacijskog sustava. Dok konceptualni, logički i fizički modeli sadrže, svaki na svojoj razini apstrakcije, podatke o podacima (metapodatke) kojima je opisana struktura stvarnih podataka u bazi podataka, baza podataka sadrži i stvarne podatke informacijskog sustava, kao i metapodatke koji opisuju podatke u bazi podataka.“

Baze podataka se primarno razlikuju po modelu na kojem su bazirane, danas postoji puno različitih modela baza podataka no i dalje su najkorištenije i najrelevantnije relacijske baze podataka. Kako bi se moglo raditi s bazama podataka potreban je sustav za upravljanje bazama podataka (engl. *Database Management System - DBMS*). To je programski sustav koji sadrži funkcije za definiranje baze podataka, funkcije za manipulaciju podacima u bazi podataka, te upravljačke funkcije poput zaštite od neovlaštenog korištenja, očuvanje integriteta baze podataka i statističko praćenje rada baze podataka. [9]

U ovom poglavlju će biti opisane baze podataka korištene u ovom radu, biti će prikazano instaliranje i kreiranje svake od baza podataka, njihovo povezivanje sa prethodno opisanom aplikacijom te operacije potrebne za kreiranje, dohvaćanje, ažuriranje i brisanje podataka. Baze podataka, točnije sustavi za upravljanje bazama podataka korišteni u ovom radu su izabrani na temelju popularnosti i modelu kako bi se prikazala razlika između najkorištenijih baza podataka, izabrane su tri relacijske baze podataka i dvije ne-relacijske koje su na vrhu popularnosti za model na kojem su bazirane. [10]

4.1. SQLServer

SQL Server je sustav za upravljanje bazama podataka kojeg je razvio Microsoft. Služi za upravljanje relacijskim bazama i sadrži širok izbor aplikacija za analitiku, obradu transakcija i poslovnu inteligenciju. SQL Server služi ne samo za aplikacije na industrijskoj razini koje su dan danas dostupne već i za akademske potrebe. Najčešće se koristi uz pomoć alata SSMS – SQL Server Management Studio koji služi za upravljanje serverom i bazama. Postoje različite verzije SQL Servera: Enterprise, Standard, Express i Developer. [11]

U ovom radu će biti korištena SQL Server Express verzija, ona je namijenjena za manje web i mobilne aplikacije koje su osnovane na podacima do 10 GB, također Express verzija je besplatna.

4.1.1. Instalacija i kreiranje

Instalacija SQL servera je korištenjem ASP.NET Core okruženja automatski obavljena kod instalacije Visual Studio modifikacije. Koristi se LocalDB, laganija verzija SQL Server Express baze podataka koja je namijenjena programerima za izgradnju aplikacija.

Kreiranje same baze podataka se također provodi automatski korištenjem Entity Framework Core-a (EF Core) za rad s bazom podataka, to je laganija, proširiva tehnologija otvorenog koda koja se koristi za mapiranje objekata i relacija u bazi. EF Core radi na principu da je prvo potrebno napisati klase modela a zatim se na temelju tih klasa kreira baza podataka. Klase koje se definiraju nisu posebno vezane za neki okvir već su napisane kao POCO objekti ili obični objekt stare klase (eng. *Plan old CLR object*). Kao što je vidljivo na slici 15 dodane su potrebne klase u folder nazvan *models*. Ovako izgleda klasa za objekt inventara:

```
5  public class Inventory
6      {
7          public int ID { get; set; }
8          public int SneakerID { get; set; }
9          public float Size { get; set; }
10         public int Quantity { get; set; }
11         public Sneaker Sneaker { get; set; }
12         public ICollection<Cart> Carts { get; set; }
13         public ICollection<OrderDetails> OrderDetails { get; set; }
14     }
```

Ovo je jedna od jednostavnijih klasa modela, sadrži podatke definirane na slici 14 kao što su *ID*, *SneakerID*, *Size* i *Quantity*. Dodana su još dva svojstva u model u linijama 12 i 13 koja predstavljaju navigacijska svojstva za bazu podataka. Jedna stavka inventara se može nalaziti u više košarica i u više detalja narudžbe, stoga se za ta svojstva definiraju kao kolekcija (`ICollection`), dok jedan inventar može biti vezan samo za jednu tenisicu stoga model

sadrži samo jedan entitet klase *Sneaker*. Uz definiranje čiste klase modela podataka moguće je uz svaki od podataka dodati razne atribute vidljive kod definiranja modela košarice:

```
5  public class Order
6      {
7          public int ID { get; set; }
8          [DisplayFormat(NullDisplayText = "Guest")]
9          public int? UserID { get; set; }
10         public string Name { get; set; }
11         [Display(Name = "Created At")]
12         public DateTime CreatedDate { get; set; }
13         [Required]
14         [Display(Name = "Payment Type")]
15         public string PaymentType { get; set; } = string.Empty;
16         public string Status { get; set; } = "pending";
17         public User User { get; set; }
18         public ICollection<OrderDetails> OrderDetails { get; set; }
19     }
```

Moguće je definirati atribute poput `[DataType]`, `[DisplayFormat]`, `[StringLenght]`, `[Column]`, `[Required]` i `[Display]`. U ovom primjeru koristi se `DisplayFormat` za ispis teksta *Guest* ako je polje *UserID* prazno. Zatim se koristi `Display` za promjenu naziva polja kod prikaza podataka i `Required` atribut za polja koja je obavezno popuniti kod ispunjavanja forme, polja koja u bazi ne mogu biti nula imaju automatski dodijeljen `Required` atribut. Na liniji 16 je također dodana automatska vrijednost za polje statusa ako nije eksplicitno definiran u kodu ili ispunjen u formi od strane korisnika.

Nakon definiranja svih potrebnih klasa modela potrebno je kreirani inicijalnu shemu baze podataka, za to će biti korištena funkcionalnost migracije Entity Frameworka. Preko navigacije Visual Studio-a potrebno je otvoriti Package Manger Console i upisati sljedeće naredbe:

```
Add-Migration InitialCreate
Update-Database
```

Prva naredba kreira inicijalnu shemu baze podataka, `InticalCreate` je samo naziv migracije i moguće ga je promijeniti tako da opisuje svrhu migracije. Naredba kreira shemu na temelju konteksta baze podataka koji će biti pobliže opisan u sljedećem poglavlju. Druga naredba izvršava kod migracija koje još nisu primijenjene, u ovom slučaju izvršit će se jedna nova migracija.

4.1.2. Povezivanje s aplikacijom

Kako bi povezali bazu podataka s aplikacijom potrebno je dodati ključ za spajanje na bazu podataka u postavke aplikacije, točnije u *appsettings.json* datoteku:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "SneakerShopContext":
       "Server=(localdb)\\mssqllocaldb;Database=SneakerShop.Data;
       Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}
```

Zatim se dodaje kontekst baze podataka u *Program.cs* datoteku pomoću sljedećeg koda:

```
builder.Services.AddDbContext<SneakerShopContext>(options =>

    options.UseSqlServer(builder.Configuration.GetConnectionString("Sneak
    erShopContext") ?? throw new InvalidOperationException("Connection
    string 'SneakerShopContext' not found.")));
```

Dodavanjem ovog konteksta ASP.NET Core sistem za konfiguraciju čita *ConnectionString* ključ i spaja se na bazu podataka. Nakon kreiranja klasa za modele podataka korištene u radu, koristi se alat za *scaffolding*, to je proces koji koristi kontekst baze podataka i model koji smo definirali kako bi se automatski generirale Razor stranice za CRUD operacije. Tokom *scaffolding* procesa generira se i dodatna klasa koja predstavlja kontekst baze podataka, onda spaja tablice u bazi sa modelima u projektu i preko te klase konteksta se izvršava komunikacija i upravljanje podacima u bazi. Ovako izgleda klasa *SneakerShopContext.cs*:

```
namespace SneakerShopSQLServer.Data
{
    public class SneakerShopContext : DbContext
    {
        public SneakerShopContext(DbContextOptions<SneakerShopContext>
         options)
            : base(options)
        {
        }

        public DbSet<User> Users { get; set; }
        public DbSet<Sneaker> Sneakers { get; set; }
        public DbSet<Inventory> Inventory { get; set; }
        public DbSet<Cart> Carts { get; set; }
        public DbSet<Order> Orders { get; set; }
```

```
22          public DbSet<OrderDetails> OrderDetails { get; set; }
23
24          protected override void OnModelCreating(ModelBuilder
        modelBuilder)
25          {
26              modelBuilder.Entity<User>().ToTable(nameof(User));
27              modelBuilder.Entity<Sneaker>().ToTable(nameof(Sneaker));
28              modelBuilder.Entity<Cart>().ToTable(nameof(Cart));
29              modelBuilder.Entity<Inventory>()
                    .ToTable(nameof(Models.Inventory));
30              modelBuilder.Entity<Order>().ToTable(nameof(Order));
31              modelBuilder.Entity<OrderDetails>()
                    .ToTable(nameof(Models.OrderDetails));
32          }
33      }
34  }
```

4.1.3.CRUD operacije

Operacije kreiranja, čitanja, ažuriranja i brisanja podataka (eng. *CRUD – Create, Read, Update, Delete*) su osnovne operacije za rad s bazom podataka. Entity Framework Core funkcionira tako da se koristi prethodno prikazana klasa konteksta koja predstavlja objekt sesije baze te se preko tog objekta izvršavaju CRUD operacije. Za generiranje upita na bazu umjesto klasičnog pisanja SQL upita u ovom se slučaju koristi jezični integrirani upit (eng. *Langueage Integrated Query – LINQ*). Za početak svaka klasa koja komunicira s bazom podataka treba u konstruktoru sadržavati prethodno spomenutu klasu konteksta:

```
13          private readonly SneakerShopContext _context;
14          private readonly ILogger<IndexModel> _logger;
15
16          public IndexModel(SneakerShopContext context, ILogger<IndexModel>
              logger)
17          {
18              _context = context;
19              _logger = logger;
20          }
```

U nastavku će biti prikazane naredbe korištene za dohvaćanje i obradu podataka vezanih uz model tenisice, svaka naredba koristi `_context` objekt iz konstruktora.

Kreiranje:

```
_context.Sneakers.Add(Sneaker);
await _context.SaveChangesAsync();
```

Prikupljanjem podataka iz forme za dodavanje tenisice se kreira objekt `Sneaker`, naredbom `Add` se počinje pratiti dani entitet a pozivom `SaveChangesAsync` naredbe će se detektirane promijene u kontekstu baze asinkrono spremiti i tenisica će biti dodana u bazu. Ova vrsta naredbe se uvijek poziva u *OnPostAsync()* metodi jer se ona izvršava predavanjem ispunjene forme i asinkrone je vrste što je potrebno za asinkrono dohvaćanje podataka.

Čitanje:

```
public IList<Sneaker> Sneaker { get; set; } = default!;
Sneaker = await _context.Sneakers.ToListAsync();
```

Ovo je primjer čitanja svih redova iz tablice tenisica, potrebno je inicijalizirati listu objekata *Sneaker* te pozvati `ToListAsync()` naredbu koja će asinkrono vratiti enumeriranu listu svih tenisica kojima je moguće pristupiti preko indexa. Ovaj oblik čitanja se koristi kod *OnGetAsync()* metode koja se izvršava kod učitavanja stranice kako bi se dohvatili podaci za prikaz u tablici ili slično. Drugi oblik dohvaćanja podataka je kada je potreban samo jedan element iz tablice kojeg tražimo na temelju određenih uvjeta. Primjer dohvaćanja jedne tenisice na temelju njezinog ID-a:

```
public Sneaker Sneaker { get; set; } = default!;
Sneaker = await _context.Sneakers.FirstOrDefaultAsync(m => m.ID == id);
```

Ažuriranje:

Ažuriranje funkcionira tako da se dohvati objekt tenisice na prethodno opisani način i s dohvaćenim podacima se ispuni forma, ukoliko korisnik mijenja podatke u formi nakon predaje forme objekt se ažurira u *OnPostAsync()* metodi te se pozivaju sljedeće naredbe za spremanje promjena u bazu:

```
_context.Attach(Sneaker).State = EntityState.Modified;
await _context.SaveChangesAsync();
```

Prvom naredbom se počinje pratiti stanje danog objekta, moguće je definirati nekoliko različitih oblika praćenja promjena, u ovom primjeru potrebno je koristiti `EntityState.Modified` jer to znači da objekt postoji u bazi i da su mu promijenjeni neki ili svi atributi. Zatim se poziva ista naredba kao i kod kreiranja zapisa, `SaveChangesAsync()` ažurira promjene koje su napravljene na danom kontekstu.

Brisanje:

```
_context.Sneakers.Remove(Sneaker);
await _context.SaveChangesAsync();
```

Za operaciju brisanja ponovno je potrebno dohvatiti objekt iz baze na prethodno prikazani način te se poziva naredba `Remove` koja se ponaša slično kao `Attach` naredba kod ažuriranja samo što sad postavlja `State` na `Deleted` te se čeka poziv sljedeće naredbe da se sam proces brisanja podataka iz baze izvrši.

4.2. MongoDB

MongoDB je baza podataka otvorenog koda (eng. *open source*) kreirana od strane MongoDB Inc. tvrtke. Jedna je od najkorištenijih baza podataka proteklih godina te je trenutno najpopularnija NoSQL baza podataka [10].

MongoDB je strukturiran tako da može sadržavati više baza podataka koje predstavljaju spremišta za kolekcije. Kolekcija je grupa dokumenata i ekvivalentna je standardnoj tablici i tako je najlakše zamisliti tu strukturu. Dokument je ekvivalentan retku racionalne baze podataka, razlika je što se on sastoji od parova ključ-vrijednost pa je stoga vrlo sličan JSON objektima. Velika prednost i poznata karakteristika MongoDB baze je što vrijednosti u dokumentu mogu primati ostale dokumente, nizove i nizove dokumenata. Prednost korištenja dokumenata je što zbog svoje strukture dokumenti odgovaraju izvornim tipovima podataka u mnogim programskim jezicima a mogućnost ugrađivanja dokumenata i korištenja nizova smanjuje potrebu za kompleksnim i skupim spajanjem tablica. Također se koristi i dinamička shema koja omogućuje kreiranje zapisa bez strogog definiranja strukture te se tako podržava polimorfizam. [12][13]

4.2.1. Instalacija i kreiranje

Za instalaciju MongoDB baze podataka potrebno je sa službenih stranica preuzeti MongoDB i MongoDB Shell. Kod instalacije MongoDB programa potrebno je locirati putanju *bin* foldera te ju dodati u *Path* sistemsku varijablu. Za instalaciju ljuske potrebno je otpakirati preuzetu datoteku na željeno mjesto te također dodati putanju do *mongosh.exe* programa u *Path* sistemsku varijablu. Za spajanje na MongoDB potrebno je odrediti mjesto gdje će se podaci spremati, za potrebe ovog rada kreiran je folder C:\SneakerShopData te se u naredbenom retku Windowsa (ne u preuzetoj ljuski) pokrene sljedeća naredba:

```
mongod --dbpath C:\SneakerShopData
```

Sljedeće je potrebno pokrenuti MongoDB ljusku direktnim pokretanjem mongosh.exe programa ili upisivanjem *monosh* naredbe u naredbeni redak. Za kreiranje same baze podataka koristi se sljedeća naredba:

```
use SneakerShop
```

Pokretanjem naredbe kreira se baza ako ne postoji i otvara se veza na nju, sljedeće je potrebno kreirati kolekcije naredbom `db.createColletion('nazivKolekcije')`. Zbog prirode MongoDB baze nije potrebno zasebno kreirati *OrderDetails* kolekciju kao što je na primjer SQL Server baza imala *OrderDetails* tablicu jer je ovdje moguće pohraniti niz *OrderDetails* objekata u jednoj narudžbi (*Order* kolekciji) što će biti prikazano i u modelu.

Za kreiranje klasa modela pojedinog objekta odnosno dokumenta uređeni su modeli korišteni kod SQL Server baze podataka. Obrisana su nepotrebna `ICollection` polja koja su kod SQL servera služila za ispravno migriranje korištenjem *scaffoldinga*. Najbitnija promjena je dodavanje dviju anotacija kod ID-a svakog modela te pretvorba svih ID-ova u znakovni tip podatka, ovdje je vidljiv primjer za inventar:

```
5   namespace SneakerShopMongoDB.Models
6   {
7       public class Inventory
8       {
9           [BsonId]
10          [BsonRepresentation(BsonType.ObjectId)]
11          public string? ID { get; set; }
12          public string? SneakerID { get; set; }
13          public float Size { get; set; }
14          public int Quantity { get; set; }
15          public Sneaker? Sneaker { get; set; }
16      }
17  }
```

Definiranje ID-a na ovaj način je potrebno zbog ispravnog mapiranja objekta u MongoDB kolekciju. `[BsonId]` anotacija označava da je to polje primarni ključ dokumenta u bazi, a `[BsonRepresentation(BsonType.ObjectId)]` dozvoljava parsiranje *ObjectId* tipa kakav je spremljen u bazi u niz znakova (eng. *string*). [14]

Ostale promjene kod modela su da sada narudžba sadrži sljedeći atribut:

```
public IList<OrderDetails> OrderDetails { get; set; }
```

Klasa za model *OrderDetails* je također izmijenjena jer više nije spremljena sama za sebe pa nema potrebe da sadrži vlastiti ID i vanjski ključ za narudžbu na koju su detalji vezani. Model detalja narudžbe stoga izgleda ovako:

```
3   namespace SneakerShopMongoDB.Models
4   {
5       public class OrderDetails
6       {
7           public string? InventoryID { get; set; }
8           public int Quantity { get; set; }
9           public Inventory Inventory { get; set; }
10      }
11  }
```

4.2.2. Povezivanje s aplikacijom

Za rad s MongoDB bazom podataka u ASP.NET Core projektu potrebno je instalirati *MongoDB.Driver* NuGet paket korištenjem *NuGet Package Managera* ili *Package Manager* konzole. Za ispravno povezivanje baze s aplikacijom potrebno je izmijeniti ili dodati tri stvari, od kojih je prva da se u *appsettings.json* datoteku doda sljedeća konfiguracija:

```
9     "SneakerShopDatabase": {
10      "ConnectionString": "mongodb://localhost:27017",
11      "DatabaseName": "SneakerShop",
12      "SneakerCollectionName": "Sneaker",
13      "InventoryCollectionName": "Inventory",
14      "OrderCollectionName": "Order",
15      "CartCollectionName": "Cart",
16      "UserCollectionName": "User"
17    }
```

Zatim je u *Models* folder uz postojeće klase modela dodana nova klasa nazvana *SneakerShopDatabaseSettings.cs* koja sadrži sljedeće:

```
5   public class SneakerShopDatabaseSettings
6       {
7           public string ConnectionString { get; set; } = null!;
8           public string DatabaseName { get; set; } = null!;
9           public string SneakerCollectionName { get; set; } = null!;
10          public string InventoryCollectionName { get; set; } = null!;
11          public string OrderCollectionName { get; set; } = null!;
12          public string CartCollectionName { get; set; } = null!;
13          public string UserCollectionName { get; set; } = null!;
14      }
```

Posljednja izmjena je u *Program.cs* klasi, gdje se umjesto koda za spajanje na prethodnu bazu podataka dodaje registriranje instance klase postavki prikazane iznad te čitanje konfiguracijskih podataka dodanih na početku:

```
7   builder.Services.Configure<SneakerShopDatabaseSettings>(
8       builder.Configuration.GetSection("SneakerShopDatabase"));
```

Sukladno klasi konteksta koja je kod SQL Server implementacije bila generirana u *scaffolding* procesu kod MongoDB baze koristi se klasa servisa koja zapravo ima istu svrhu. Može se koristiti isti proces da se automatski generira ali bi trebalo mijenjati gotovo svaku generiranu naredbu, dodatno zbog očuvanja logike aplikacije koristit će se postojeći kod iz aplikacije bazirane na SQL Server-u. Stoga je potrebno napraviti promjenu na novu bazu tako da se obriše stara klasa konteksta i doda se ne novi folder *Services* koji će sadržavati klasu *SneakerShopService.cs*. U novoj klasi se definiraju kolekcije koje se nalaze u bazi te u konstruktor čita prethodno definiranu konfiguraciju za spajanje na bazu podataka. Dio klase *SneakerShopService* izgleda ovako:

```
8   namespace SneakerShopMongoDB.Services
9   {
10      public class SneakerShopService
11      {
```

```
12          private readonly IMongoCollection<Sneaker> Sneakers;
13          private readonly IMongoCollection<Inventory> Inventories;
14          private readonly IMongoCollection<Order> Orders;
15          private readonly IMongoCollection<Cart> Carts;
16          private readonly IMongoCollection<User> Users;
17
18          public SneakerShopService(IOptions<SneakerShopDatabaseSettings>
             settings) {
19              var mongoClient = new
                   MongoClient(settings.Value.ConnectionString);
20              var mongoDatabase =
                   mongoClient.GetDatabase(settings.Value.DatabaseName);
21
22              Sneakers = mongoDatabase.GetCollection<Sneaker>(
                   settings.Value.SneakerCollectionName);
23              Inventories = mongoDatabase.GetCollection<Inventory>(
                   settings.Value.InventoryCollectionName);
24              Orders = mongoDatabase.GetCollection<Order>(
                   settings.Value.OrderCollectionName);
25              Carts = mongoDatabase.GetCollection<Cart>(
                   settings.Value.CartCollectionName);
26              Users = mongoDatabase.GetCollection<User>(
                   settings.Value.UserCollectionName);
27          }
28          /*CRUD OPERACIJE*/
             …
208     }
209 }
```

U prikazanom kodu instanca klase *SneakerShopDatabaseSettings* je dobivena injektiranjem ovisnosti (eng. dependency injection) te na taj način može čitati vrijednosti konfiguraciju prethodno dodane u *appsettings.json* [14]. Nakon dodavanja prikazane klase, potrebno je registrirati klasu sa injektiranjem ovisnosti dodavanjem sljedećeg koda u *Program.cs* klasu:

```
builder.Services.AddSingleton<SneakerShopService>();
```

Prema službenim uputama *MongoClient* bi trebao biti registriran kao singleton, a u ovom slučaju *SneakerShopService* uzima direktnu ovisnost nad klijentom. [14]

4.2.3.CRUD operacije

Operacije za kreiranje, čitanje, ažuriranje i brisanje su kod implementaciji MongoDB baze malo drugačije postavljene nego kod prethodno prikazanih operacija vezanih uz SQL Server. U ovom su slučaju operacije definirane u klasi *SneakerShopService* prikazanoj iznad. Operacije potrebne za svaki model su grupirane zajedno te se funkcije pozivaju preko instance klase servisa. Ovako izgledaju sve funkcije i operacije vezane uz model tenisice:

```
29  public async Task<List<Sneaker>> GetSneakerAsync() =>
30      await Sneakers.Find(_ => true).ToListAsync();
31
32  public async Task<Sneaker?> GetSneakerAsync(string id) =>
33      await Sneakers.Find(x => x.ID == id).FirstOrDefaultAsync();
34
```

```
35  public async Task CreateSneakerAsync(Sneaker newSneaker) =>
36      await Sneakers.InsertOneAsync(newSneaker);
37
38  public async Task UpdateSneakerAsync(string id, Sneaker updatedSneaker)
39      => await Sneakers.ReplaceOneAsync(x => x.ID == id, updatedSneaker);
40
41  public async Task RemoveSneakerAsync(string id) =>
42      await Sneakers.DeleteOneAsync(x => x.ID == id);
```

Kako bi stranice aplikacije koristile prikazane funkcije iz klase servisa potrebno je dodati klasu u konstruktor svake stranice:

```
12        private readonly SneakerShopService _sneakerShopService;
13        private readonly ILogger<IndexModel> _logger;
14
15      public IndexModel(SneakerShopService sneakerShopService,
            ILogger<IndexModel> logger)
16        {
17            _sneakerShopService = sneakerShopService;
18            _logger = logger;
19        }
```

U nastavku će biti prikazane operacije, odnosno korištenje funkcija klase servisa uz pomoć `_sneakerShopService` objekta zajedno sa pojašnjenjem CRUD operacija.

Kreiranje:

```
await _sneakerShopService.CreateSneakerAsync(Sneaker);
```

Poziva se funkcija `CreateSneakerAsync` koja nad kolekcijom tenisica izvršava naredbu `InsertOneAsync()` te se tako dodaje novi dokument u bazu podataka.

Čitanje:

```
public IList<Sneaker> Sneaker { get; set; } = default!;
Sneaker = await _sneakerShopService.GetSneakerAsync();
```

Operacija čitanja poziva funkciju koja nad kolekcijom izvršava naredbu `Find(_ => true).ToListAsync()` koja vraća listu svih dokumenata iz kolekcije. Koristi se i zasebna naredba za dohvaćanje pojedinih dokumenta kod koje se stavlja željeni uvjet u `Find()` naredbu i umjesto `ToListAsync()` koristi se `FirstOrDefaultAsync()`:

```
public Sneaker Sneaker { get; set; } = default!;
Sneaker = await _sneakerShopService.GetSneakerAsync(id);
```

Ažuriranje:

```
await _sneakerShopService.UpdateSneakerAsync(Sneaker.ID, Sneaker);
```

Ažuriranje za razliku od operacije u SQL Server bazi mijenja cijeli dokument novim koristeći naredbu `ReplaceOneAsync(filter, replacement)`.

Brisanje:

```
await _sneakerShopService.RemoveSneakerAsync(id);
```

4.3. Cassandra

Cassandra je distribuirana baza podataka otvorenog koda proizvedena od strane Apache Software tvrtke. Karakteristike ove baze podataka su da je decentralizirana, elastično skalabilna, široko dostupna, prilagodljivo konzistentna, otporna na greške i stupno orijentirana. Da je baza distribuirana znači da može biti pokrenuta na više računala a u isto vrijeme korisniku djeluje kao jedna cjelina. Moguće je pokrenuti Cassandru na jednom čvoru no maksimalni potencijal ove baze podataka je vidljiv kada je pokrenuta na više računala. Distribuirani dizajn je temeljen na Amazon-ovoj Dynamo bazi podataka a model podataka Cassandre uzima inspiraciju od Google-ove Bigtable širokostupne (eng. *wide-column*) baze podataka. [15][16]

Trenutno je Cassandra jedna od najpopularnijih baza podataka [10], te se koristi na puno poznatih Internet stranica. Odabrao sam ovu bazu za usporedbu u ovom radu kako bi predstavio još jednu ne-racionalnu bazu podataka koja je drastično drugačija od klasične implementacije tablica te kako bi prikazao sličnosti i razlike. U ovom poglavlju će biti ukratko prikazana instalacija Cassandre, kreiranje baze te promjene koje su učinjene u aplikaciji i modelu kako bi aplikacija radila koristeći ovu bazu podataka.

4.3.1. Instalacija i kreiranje

Za pokretanje Cassandra baze podataka na Windows operacijskom sustavu potrebno je instalirati Javu i Python. Verzije korištene u ovom radu su Java Development Kit 8u251, Python 2.7 i Cassandra 3.11.15. [17]

Nakon preuzimanja potrebnih datoteka sa službenih izvora i izvršavanja instalacije potrebno je za svaki od navedenih programa dodati sistemsku varijablu:

1. JAVA_HOME: C:\Program Files\Java\jdk1.8.0_251
2. Path > New: C:\Python27
3. CASSANDRA_HOME: C:\Cassandra\apache-cassandra-3.11.15
4. Path > New: C:\Cassandra\apache-cassandra-3.11.15\bin

Nakon ispravnog instaliranja svih programa i postavljanja sistemskih varijabli Cassandra server se pokreće tako da se otvori naredbeni redak u direktoriju 4. koraka te se pokrene naredba `cassandra`. Potrebno je ostaviti taj naredbeni redak otvoren tako da server radi u pozadini i otvoriti novi sa iste lokacije i izvršiti naredbu `cqlsh` koja pokreće Cassandra ljusku (eng. *shell*) preko koje možemo kreirati bazu podataka i izvršavati komande na pokrenutom serveru. [17]

Baza podataka, točnije *keyspace* kod Cassandre se kreira izvršavanjem sljedeće naredbe unutar cqlsh-a:

```
CREATE KEYSPACE SneakerShop WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'}
```

Koristi se `SimpleStrategy` replikacija podataka jer se u implementaciji za ovaj rad koristi samo jedan lokalni čvor. Ukoliko bih koristio više čvorova potrebno je staviti `NetworkTopologyStrategy`. [18]

Nakon kreiranja baze potrebno ju je odabrati naredbom `use SneakerShop;` te zatim može početi kreiranje tablica. Kod dizajniranja modela za Cassandra bazu podataka potrebno je imati na umu ključne razlike u usporedbi s racionalnim bazama. Cassandra nema mogućnost spajanja (eng. *join*) tablica u upitu, ne prakticira se korištenje vanjskih ključeva, iako se mogu spremiti u tablicu ne koriste se za operacije poput kaskadnog brisanja, takve se operacije moraju implementirati u samoj aplikaciji. Cassandra je osmišljena i nudi najbolje performanse kada se model dizajnira oko upita koji su potrebni u aplikaciji, no budući da je to više aplikacijski zahtjevno te bi zahtijevalo puno promjena u logici same aplikacije što bi u konačnici utjecalo i na usporedivost brzina za ovaj rad, modeli podataka će ostati što sličniji onima za relacijske baze. Bitna razlika je što će biti korišteni korisnički definirani tipovi podataka (eng. *user-defined types – UDTs*) koji omogućuju da se ugnježđuju podaci, stoga će za svaku tablicu (model) koji sadrži vanjski ključ biti dodano i polje koje će sadržavati korisnički definirani tip sukladan modelu koji se referencira vanjskim ključem. Ovim pristupom će biti riješeno povezivanje podataka bez definiranja novih tablica a budući da Cassandra podržava i potiče dupliciranje podataka ukoliko je potrebno bez velikog utjecaja na performanse smatram da je ovo najprikladniji pristup za implementaciju Cassandre bez drastičnom mijenjanja logike aplikacije. U nastavku će biti prikazano kreiranje jedne tablice i korisnički definiranog tipa podataka kojeg tablica koristi.

```
CREATE TYPE IF NOT EXISTS SneakerShop.Sneaker (
ID UUID,
Brand text,
Name text,
Description text,
Picture1 blob,
Picture2 blob,
Price decimal
);
```

```
CREATE TABLE IF NOT EXISTS SneakerShop.Inventory (
ID UUID PRIMARY KEY,
SneakerID UUID,
Size float,
Quantity int,
Sneaker frozen<Sneaker>
);
```

Prvo je potrebno kreirati korisnički definirati tip kako bi se on mogao koristiti u kasnije kreiranoj tablici. Jedina razlika kod kreiranja tipa i tablice iste vrste je što korisnički definirani tip ne sadrži primarni ključ. Sljedeće su vidljivi neobični tipovi podataka specifični za Cassandru, kao prvo Cassandra ne podržava standardno automatsko povećavanje vrijednosti primarnog ključa, iako postoje brojači s kojima se može takva funkcionalnost simulirati, generalno se preporuča korištenje univerzalno jedinstvenog identifikatora (eng. *Universally unique identifier – UUID*) koji predstavlja 128 bitni unikatni niz znakova određenog formata. Kod korištenja korisnički definiranog tipa *Sneaker* u tablici *Inventory* preporuča se korištenje `frozen<>` atributa koji označuje da se podatak ne može uređivati već se mora cijeli zamijeniti ukoliko je potrebno napraviti promijene. [19]

Nakon kreiranja tablica u bazi potrebno je ažurirati modele podataka u aplikaciji kako bi bili prikladni za Cassandru. U nastavku će biti prikazan model za objekt inventara kako bi se vidjele promjene:

```
6     [Table("sneakershop.inventory")]
7     public class Inventory
8     {
9           public Guid ID { get; set; } = Guid.NewGuid();
10          public Guid SneakerID { get; set; }
11          public float Size { get; set; }
12          public int Quantity { get; set; }
13          public Sneaker Sneaker { get; set; }
14    }
```

`[Table]` anotacija se koristi za mapiranje klase sa tablicom u bazi te za korisnički definirane tipove. Promijenjeni su i tipovi podataka u modelima, umjesto niza znakova ili brojeva kod ID-ova, sada se koristi *Guid* (*Globally unique identifier*), što je Microsoftova verzija prije spomenutog UUID-a, oba tipa su međusobno kompatibilna. Modeli su napisati tako da se novi ID generira kod inicijaliziranja objekta tako se ne mora eksplicitno pozivati `Guid.NewGuid()` kod svakog kreiranja. Naredba se poziva samo kada je potrebno da se zna vrijednost ID-a za zapis u drugu tablicu kao vanjski ključ, na primjer ID detalja narudžbe u tablicu narudžbe.

4.3.2. Povezivanje s aplikacijom

Nakon izmjene modela podataka korištenih sa Cassandra bazom podataka potrebno je prvo instalirati NuGet paket *CassandraCSharpDriver* koji nudi modernu i podesivu C# knjižnicu za Apache Cassandru i DataStax Enterprise. Sljedeće je potrebno u *appsettings.json* datoteku dodati kod konfiguracije spajanja na bazu:

```
9     "Cassandra": {
10      "ContactPoints": "localhost",
11      "Keyspace": "sneakershop"
12    }
```

Nakon dodavanja postavki može se urediti klasa *SneakerShopContext.cs* prethodno korištena kod implementacije SQL Server-a. Zapravo je samo naziv klase isti a sve ostalo se mijenja kako bi se omogućio rad s novom bazom, ta klasa sad izgleda ovako:

```
8     public class SneakerShopContext
9     {
10          private readonly Cluster _cluster;
11          private readonly Cassandra.ISession _session;
12          private readonly Mapper _mapper;
13
14          public SneakerShopContext(IConfiguration configuration)
15          {
16                var contactPoints =
                              configuration["Cassandra:ContactPoints"];
17                var keyspace = configuration["Cassandra:Keyspace"];
18
19                _cluster = Cluster.Builder()
20                      .AddContactPoints(contactPoints.Split(','))
21                      .Build();
22
23                _session = _cluster.Connect(keyspace);
24                _mapper = new Mapper(_session);
25
26                _session.UserDefinedTypes.Define(
27                      UdtMap.For<Inventory>(),
28                      UdtMap.For<Sneaker>(),
29                      UdtMap.For<Orders>(),
30                      UdtMap.For<User>(),
31                      UdtMap.For<OrderDetails>()
32                );
33          }
34
35          public Cassandra.ISession GetSession()
36          {
37                return _session;
38          }
39
40          public Mapper GetMapper()
41          {
42                return _mapper;
43          }
            …
69    }
```

U prikazanom kodu se čitaju konfiguracijski podaci kako bi se uspostavila veza na bazu, zatim se na temelju kreirane sesije stvara `Mapper` objekt koji se koristi za izvršavanje svih potrebnih operacija na bazu. Dodatno se definiraju i korisnički definirani tipovi tako da se povežu s klasama modela podataka kao što je vidljivo u linijama 26-32.

Sljedeće je potrebno dodati novo izmijenjenu klasu kao singleton servis u *Program.cs* klasi dodavanjem sljedeće linije koda:

```
builder.Services.AddSingleton<SneakerShopContext>();
```

4.3.3. CRUD operacije

Operacije za kreiranje, čitanje, ažuriranje i brisanje su u ovom slučaju definirane na dva mjesta u kodu. Operacije za čitanje svih zapisa svake od tablica u bazi su definirane u klasi konteksta prikazanoj iznad, između linija 43 i 69. Jedna od funkcija, na primjer za dohvaćanje svih tenisica, izgleda ovako:

```
45          public async Task<List<Sneaker>> GetAllSneakers()
46          {
47                var sneakers= await _mapper.FetchAsync<Sneaker>();
48                return sneakers.ToList();
49          }
```

Koristi se naredba `FetchAsync<>()` koja vraća *IEnumerable* listu iz baze te se naredbom `.ToList()` lista pretvara u oblik potreban u aplikaciji.

Sve ostale operacije za komunikaciju s bazom podatak se definiraju kod svake stranice kao što je bilo i kod SQL Server implementacije. Svaka stranica treba također u konstruktoru inicijalizirati klasu konteksta kako bi se moglo pristupiti samom kontekstu da se pozivaju naredbe poput gore navedene ili da se pristupi *mapper* objektu za pisanje svih ostalih operacija:

```
14          private readonly SneakerShopContext _context;
15          private readonly ILogger<IndexModel> _logger;
16          public IndexModel(SneakerShopContext context,
                ILogger<IndexModel> logger)
17          {
18              _context = context;
19              _logger = logger;
20          }
```

Ili:

```
19          private readonly ILogger<CreateModel> _logger;
20          private readonly Mapper _mapper;
21          public CreateModel(SneakerShopContext context,
                ILogger<CreateModel> logger)
22          {
23              _logger = logger;
24              _mapper = context.GetMapper();
25          }
```

U nastavku će biti prikazane operacije vezane uz bazu podataka korištene za tablicu tenisica.

Kreiranje:

```
public Sneaker Sneaker { get; set; } = default!;
await _mapper.InsertAsync(Sneaker);
```

Inicijaliziran je objekt tenisice, koji je u klasi anotiran tako da se objekt povezan sa tablicom u bazi podataka i samo je potrebno pozvati `InsertAsync()` naredbu, unutar *OnGetAsync()* metode, koja upisuje dani POCO objekt u bazu.

Čitanje:

```
public List<Sneaker> Sneaker { get; set; } = default!;
Sneaker = await _context.GetAllSneakers();
```

Za čitanje svih podataka iz tablice se koristi prije prikazana funkcija iz klase konteksta, funkcija se poziva u *OnGetAsync()* metodi jer se podaci koriste za prikaz kod učitavanja stranice. Drugi oblik čitanja podataka je dohvaćanje specifičnih redaka iz tablice, najčešće je to slučaj filtriranja po ID-u, što je kod Cassandra baze podataka u redu jer kada je tablica napisana kao u ovom radu tada primarni ključ ujedno predstavlja i ključ particije. No kada se želi filtrirati po ne indeksiranom polju tablice tada se mora koristiti ALLOW FILTERING na kraju upita, ovo se po službenim uputama treba izbjegavati jer opterećuje bazu podataka te uzdržava potencijalne performanse, no u ovom radu se dozvoljava filtriranje samo na nekolicini upita kako bih se vidio utjecaj na brzinu. Primjer dvaju različitih čitanja iz baze korištenjem *mappera*:

```
var sneaker = await _mapper.FirstOrDefaultAsync<Sneaker>("SELECT * FROM
      sneaker WHERE id = ?", id);
var inventory = await _mapper.FetchAsync<Inventory>("SELECT * FROM
      inventory WHERE sneakerID = ? AND quantity > 0 ALLOW FILTERING", id);
```

Ažuriranje:

```
await _mapper.UpdateAsync<Sneaker>("SET brand = ?, name = ?, " +
            "description = ?, picture1 = ?, picture2 = ?, price = ? " +
            "WHERE id = ? IF EXISTS",
            Sneaker.Brand, Sneaker.Name, Sneaker.Description,
            Sneaker.Picture1, Sneaker.Picture2, Sneaker.Price, Sneaker.ID);
```

Kod operacije ažuriranja naredba `UpdateAsync` dodaje `"UPDATE tablename"` dio koji nedostaje u upitu te pri tome stavlja ispravan naziv tablice pomoću klase modela za taj objekt.

Brisanje:

```
await _mapper.DeleteAsync<Sneaker>("WHERE id = ? IF EXISTS", id);
```

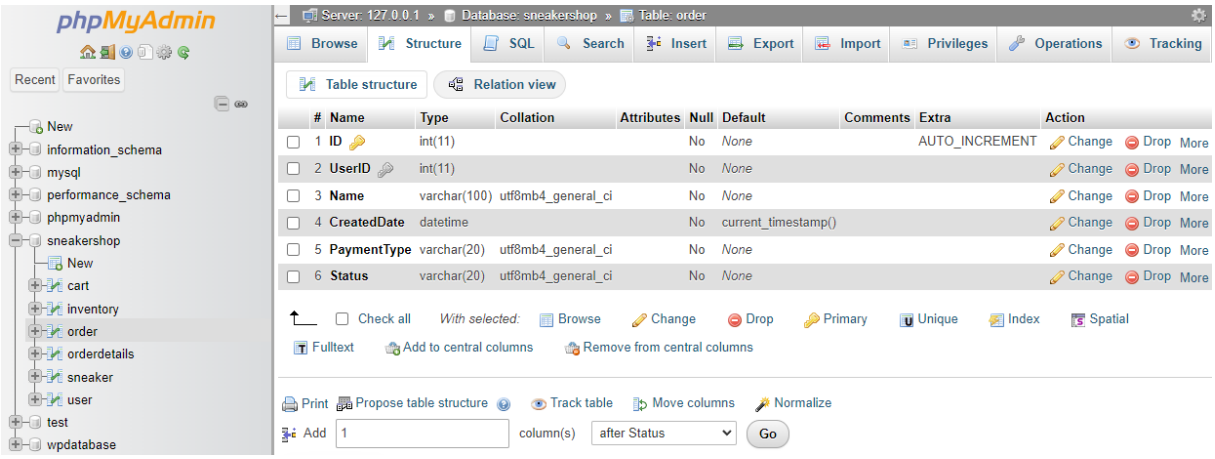
Kod brisanja naredba dodaje `"DELETE FROM tablename"` dio te izvršava upit na bazi.

4.4. MySQL

MySQL je najpopularnija baza podataka otvorenog koda, te druga najpopularnija baza podataka danas [12]. MySQL je proizveden, distribuiran i podržan od strane Oracle korporacije. Ova baza podataka je vrlo poznata po lakoći korištenja i vjerojatno je jedna od prvih baza podataka sa kojim se učenici ili studenti susreću tokom obrazovanja. Sama instalacija je brza a upravljanje bazom je jednostavno i lako razumljivo. Pouzdanost je isto jedna od vrlina MySQL-a budući da je jedna od najkorištenijih baza podataka, te se koristi i testira već više od 25 godina. MySQL se koristi u raznim scenarijima, najčešće su to aplikacije u oblaku (eng. *Cloud*), za e-trgovine kao što su Shopify, Uber i Booking.com, za socijalne platforme poput Facebook-a, Twitter-a i LinkedIn-a te za brojne lokalne aplikacije i softver kao usluga (eng. *Software as a service*) aplikacije. [20][21]

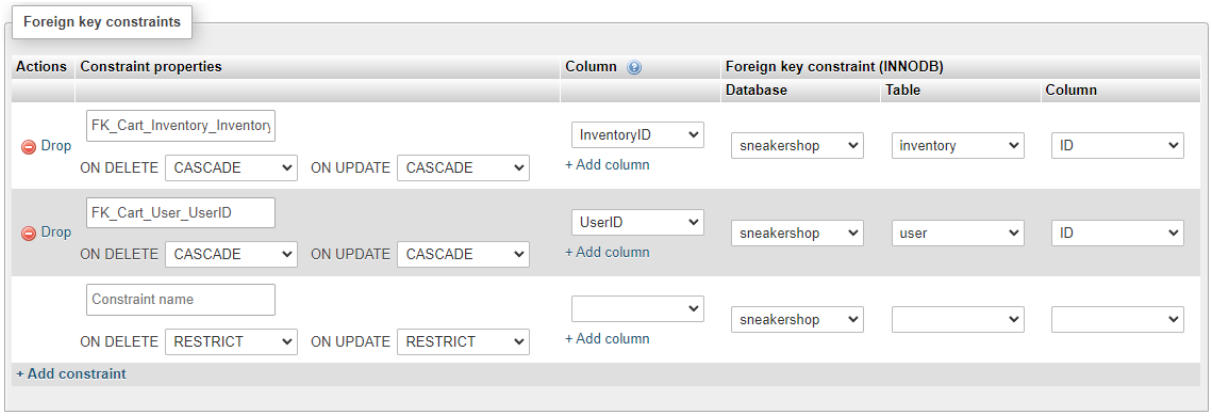
4.4.1. Instalacija i kreiranje

Za instalaciju MySQL baze podataka u ovom će radu biti korišten XAMPP program, on je besplatni višeplatformski paket rješenja web poslužitelja razvijen od strane Apache Friends projekta. Nakon preuzimanja instalacijske datoteke sa službene stranice i pokretanja instalacije mogu se ostaviti odabrane sve označene opcije za instalaciju no u ovom će slučaju biti korišten MySQL Server i phpMyAdmin program za administraciju baze preko web preglednika. Nakon uspješne instalacije potrebno je pokrenuti XAMPP Control Panel i unutar programa pokrenuti MySQL Server, upisivanjem adrese *http://localhost/phpmyadmin/* u preglednik ili klikom na *Admin* gumb unutar XAMPP-a otvara se phpMyAdmin program za grafičko administriranje bazom podataka.



Slika 16: phpMyAdmin sučelje, prikaz kreiranih tablica i struktura tablice order (vlastita izrada)

Grafičko sučelje je vrlo intuitivno za korištenje kao što je već spomenuto pa smatram da nije potrebno u detalje objašnjavati proces kreiranja nove baze podataka i pojedinih tablica. Istaknuo bih jedino tipove podataka koji se koriste kod MySQL-a budući da se razlikuju od dosada korištenih. Prva razlika, koja je vidljiva i na slici 16 je to da su brojčani i znakovni tipovi podataka ograničeni duljinom znakova koje polje može primiti. Zatim spremanje slika u tablici tenisice je do sad bilo definirano kao binarni veliki objekt (eng. *Blob – Binary large object*) ili niz bajtova, ovdje se koristi mediumblob, postoji 4 vrste binarno velikih objekata u MySQL-u a to su tinyblob (255 B), blob (64 KB), mediumblod (16MB) i longblob (4GB). Smatram da je mediumblob najprikladnija vrsta podatka za spremanje slika korištenih u ovoj aplikaciji. I posljednja razlika je što se decimalni tipovi definiraju tako da se odredi njihova preciznost i skala, pa je na primjer cijena tenisice definirana kao *decimal (18,2)* što znači da se uvijek prikazuju dvije decimale a ukupna duljina cijene može biti do 18 znamenki. Definiranje vanjskih ključeva korištenjem phpMyAdmina na primjeru tablice košarice izgleda ovako:



Slika 17: Prikaz vanjskih ključeva tablice Cart (vlastita izrada)

Na slici iznad vidljivo je na koju tablicu i stupac se vanjski ključevi referenciju te da je odabrano da se brisanjem i ažuriranjem tablice kaskadno brišu i ažuriraju povezane tablice.

Nakon uspješnog kreiranja baze podataka i potrebnih tablica potrebno je kreirati klase modela u aplikaciji, no budući da se koristi SQL Server implementacija kao gotovo rješenje za izmjenu na ostale baze podataka u ovom slučaju nema promjena u modelima već su identični onima koje koristi i SQL Server verzija aplikacije.

4.4.2. Povezivanje s aplikacijom

Za povezivanje MySQL baze podataka sa aplikacijom potrebno je u *appsettings.json* datoteku dodati sljedeću konfiguraciju:

```
9   "ConnectionStrings": {
10    "SneakerShopContext":
          "Server=localhost;Port=3306;User=root;Database=sneakershop"
11  }
```

Budući da je server instaliran lokalno stavlja se `localhost`, broj porta je vidljiv unutar XAMPP Control Panel-a te je standardno 3306, korisničko ime je `root` i naziv baze je onaj koji je definiran preko phpMyAdmin programa.

Nakon dodavanja konfiguracijskih podataka u *Program.cs* klasu se dodaje kontekst baze podataka sljedećim kodom:

```
7 builder.Services.AddDbContext<SneakerShopContext>(options =>
8     options.UseMySql(builder.Configuration
        .GetConnectionString("SneakerShopContext"),
      ServerVersion.AutoDetect(builder.Configuration
        .GetConnectionString("SneakerShopContext"))));
```

SneakerShopContext.cs klasu nije potrebno mijenjati već se koristi identičan kod kao i kod implementacije SQL Server baze podataka. Također sve klase modela podataka i CRUD operacije za komunikaciju s bazom ostaju iste kao što su prikazane u poglavlju 4.1.3.

4.5. PostgreSQL

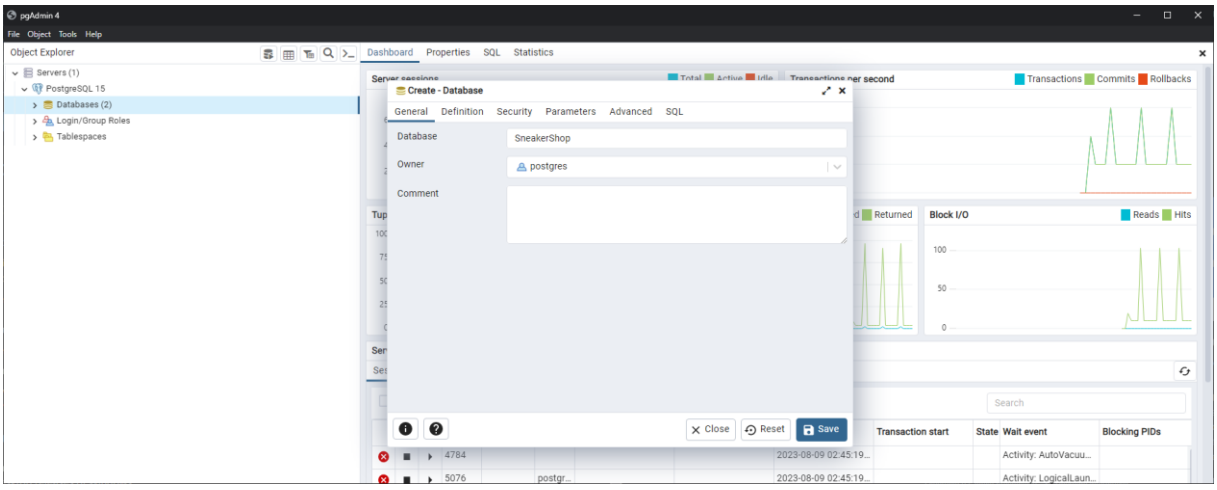
PostgreSQL je jedna od najuspješnijih relacijskih baza podataka, otvorenog je koda, puna korisnih svojstava i besplatna za korištenje. Početak razvoja ove baze seže u već sad davne osamdesete kada je baza podataka kreirana kao dio Postgres projekta na Berkeley sveučilištu što znači da se platforma aktivno razvija više od 35 godina. Dan danas se i dalje razvija pod timom programera i suradnika te se često smatra najnaprednijom bazom podataka otvorenog koda u svijetu. Kroz godine PostgreSQL je stekao vrlo dobru reputaciju zbog svoje arhitekture, pouzdanosti, integriteta podataka, robusnog seta funkcionalnosti i proširivosti. [22][23]

Model licenciranja dozvoljava komercijalno korištenje bez ograničavanja stoga brojne tvrtke nude komercijalnu potporu za PostgreSQL. Podržano je pokretanje na većini modernih operacijskih sustava te postoje brojni interaktivni instalateri, također postoje brojni alati i proširenja za upravljanje i praćenje PostgreSQL servera poput pgAdmin-a koji će biti korišten i u ovom radu. [2]

4.5.1. Instalacija i kreiranje

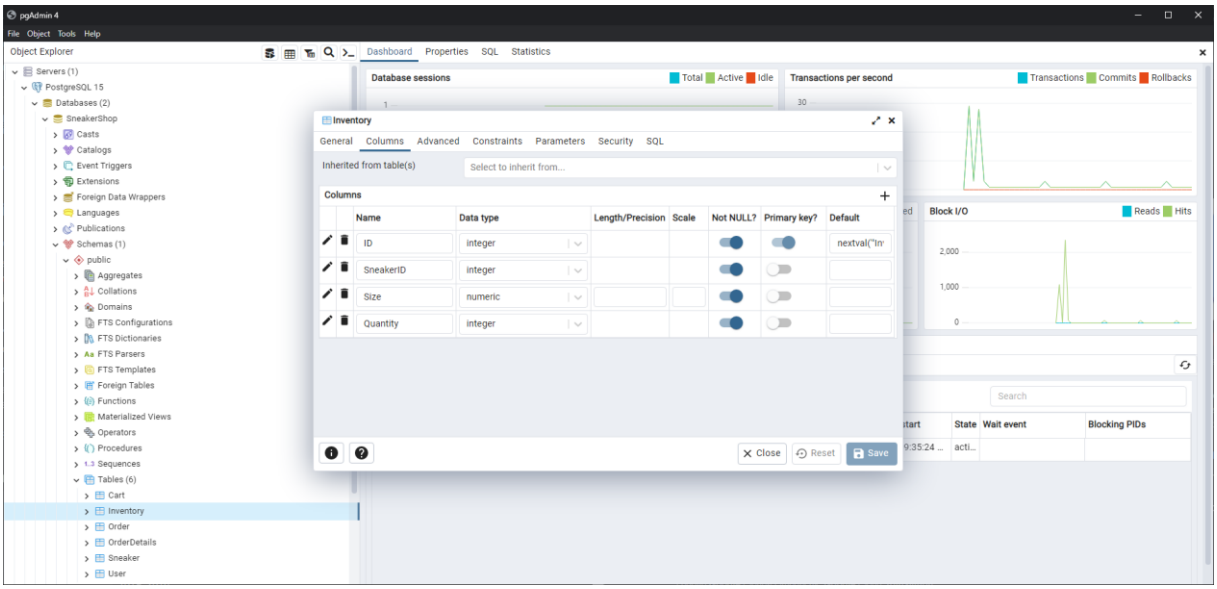
Za instalaciju PostgreSQL-a koristiti će se Windows interaktivni instalater certificiran od strance EDB tvrtke. On uključuje PostgreSQL server, pgAdmin grafičko sučelje za kreiranje i upravljanje bazom podataka te StackBuilder upravitelj paketima za instalaciju dodatnih alata i drivera. Kod instalacije se u jednom koraku odabire lozinka, bitno ju je zapamtiti jer će se koristiti kod pokretanja pgAdmin 4 alata te u aplikaciji kod spajanja na bazu.

Nakon što su programi uspješno instalirani potrebno je pokrenuti pgAdmin 4 alat te kreirati novu bazu podataka korištenjem grafičkog sučelja:



Slika 18: pgAdmin 4, kreiranje nove baze podataka (vlastita izrada)

Nakon kreiranja baze, potrebno je kreirati tablice, također korištenjem jednostavnog grafičkog sučelja, kreirane tablice su vidljive s lijeve strane programa u hijerarhiji nove baze na putanji *SnekaerShop* > *Schemas* > *public* > *Tables*. Tablice i sučelje za kreiranje tablice su prikazani na sljedećoj slici:



Slika 19: pgAdmin 4, kreirane tablice i sučelje za kreiranje tablice (vlastita izrada)

Primarni i vanjski ključevi se dodaju na kartici *Constraints* unutar prozora vidljivog na slici 19 te je odabrano da se kod ažuriranja i brisanja podataka kaskadno brišu i ažuriraju povezane tablice.

Nakon ispravnog kreiranja tablica u bazi podataka potrebno je pripremiti aplikaciju za spajanje na bazu. U ovom slučaju također se koristi SQL Server verzija aplikacije kao baza za implementaciju PostgreSQL-a te isto kao i kod MySQL implementacije nije bilo potrebno raditi promijene jer klase modela podataka, klasa konteksta *SneakerShopContext.cs* te sve CRUD operacije na bazu podataka mogu ostati iste i funkcioniraju jednako kod svih racionalnih baza podataka korištenih u ovom radu.

4.5.2. Povezivanje s aplikacijom

Kako bi ASP.NET Core aplikacija mogla raditi sa PostgreSQL bazom podataka potrebno je u VisualStudio projekt instalirati dva NuGet paketa, a to su *Npgsql.EntitiyFrameworkCore.PostgreSQL* i *Npgsql.EntityFrameworkCore.PostgreSQL.Design*. Sljedeće je potrebno urediti konfiguracijske podatke i servis za korištenje ispravne baze podataka. U *appsettings.cs* datoteku treba dodati sljedeći kod:

```
9  "ConnectionStrings": {
10     "SneakerShopContext": "Server=localhost;Database=SneakerShop;
           User ID=postgres;Password=admin"
11 }
```

U konfiguracijskim podacima se stavlja server `localhost` jer je instaliran lokalno na računalo, zatim naziv baze prethodno kreirane u pgAdmin alatu, zadani korisnički ID je `postgres` a lozinka ona koja je odabrana kod instalacije PostgreSQL-a.

Posljednja promjena je dodavanje koda za konfiguriranje konteksta kako bi se spojio na PostgreSQL bazu, u *Program.cs* klasu je potrebno staviti sljedeću naredbu:

```
8 builder.Services.AddDbContext<SneakerShopContext>(options =>
9     options.UseNpgsql(builder.Configuration
      .GetConnectionString("SneakerShopContext") ??
      throw new InvalidOperationException("Connection string
       'SneakerShopContext' not found.")));
```

5. Mjerenje brzine aplikacije

Za analizu utjecaja odabira baze podataka na brzinu aplikacije potrebno je na neki način mjeriti performanse. Baza podataka se odabire za određeni projekt ne samo na temelju poznavanja rada u određenoj okolini ili jednostavnošću korištenja i implementacije već na temelju brzine odaziva i izvršavanja operacija budući da je brzina aplikacije jedan od glavnih faktora koji krajnji korisnici primijete. U ovom poglavlju će biti pobliže pojašnjena implementacija mjerenja brzine aplikacije, biti će prikazani i objašnjeni testni podaci te procedura korištena za testiranje svake od baza podataka, za kraj će biti prikazano kako su izmjeni podaci obrađeni za korištenje u završnoj analizi.

5.1. Implementacija mjerenja

Kao što je navedeno ranije prvo je do kraja završena SQL Server verzija aplikacije, s time se podrazumijeva i implementacija mjerenja i spremanje izmjerenih podataka. Kopiranjem tog prvog projekta za implementacije ostalih baza podataka osigurava se da se mjerenje izvršava na isti način te da se mjere isti podaci budući da se taj dio ne mijenja kod promijene baze podataka.

Za samo mjerenje izvršavanja određene aktivnosti u aplikaciji koristi se *Stopwatch* klasa. Ova klasa pruža set metoda za precizno mjerenje proteklog vremena. Kao i prava štoperica klasa ima dvije glavne metode, start i stop, te je zbog toga vrlo intuitivno za korištenje. Kada želimo dobiti izmjereno proteklo vrijeme poziva se metoda *ElapsedMilliseconds* koja vraća ukupno proteklo vrijeme trenutne instance štoperice u milisekundama [25]. Primjer inicijalizacije, pokretanja, zaustavljanja i dohvaćanja vrijednosti štoperice:

```
Stopwatch stopwatch = new Stopwatch();
stopwatch.Start();
      …
stopwatch.Stop();
stopwatch.ElapsedMilliseconds
```

Prikazani primjer koda ne sprema nigdje izmjereno vrijeme, stoga trebamo dodati bilježenje ili spremanje (eng. *logging*) podataka. ASP.NET Core ne uključuje pružatelja za zapisivanje podataka (dnevnika) u datoteke, već se preporuča korištenje rješenja treće strane preuzimanjem nekih od preporučenih NuGet paketa. Stoga će se u ovom radu koristiti Serilog ekstenzija pa je potrebno instalirati *Serilog.Exstensions.Logging.File* NuGet paket. Kako bi se moglo početi zapisivati u datoteku potrebno je dodati još dvije stvari, u *Program.cs* datoteku treba dodati sljedeću liniju koda:

```
builder.Logging.AddFile("Logs/SneakerShopLogs-{Date}.txt");
```

Prethodna linija koda stvara tekstualnu datoteku u *Logs* folderu u koju će se zapisivati podaci. Sljedeće je potrebno dodati konfiguracijske podatke za bilježenje u datoteku *appsettings.json*:

```
2  "Logging": {
3     "LogLevel": {
4        "Default": "Information",
5        "Microsoft.AspNetCore": "Warning"
6     }
7  }
```

`Information` i `Warning` razine bilježenja označuju ozbiljnost zapisa koji se zapisuje, postoji 6 kategorija koje su redom: praćenje (eng. *Trace*), otklanjanje pogrešaka (eng. *Debug*), informacija (eng. *Information*), upozorenje (eng. *Warning*), pogreška (eng. *Error*), kritično (eng. *Critical*) te ništa (eng. *None*). Kada je odabrana jedna on razina ozbiljnosti to označava da će zapisivati svi podaci od te razine na više. [26]

U nastavku će biti prikazana kompletna implementacija mjerenja brzine na primjeru klase modela za stranicu kreiranja tenisice:

```
12 namespace SneakerShopSQLServer.Pages.Sneakers
13 {
14     public class CreateModel : PageModel
15     {
16         private readonly SneakerShopContext _context;
17         private readonly ILogger<CreateModel> _logger;
18         public IFormFile? imageFile1;
19         public IFormFile? imageFile2;
20 
21         public CreateModel(SneakerShopContext context,
            ILogger<CreateModel> logger)
22         {
23             _context = context;
24             _logger = logger;
25         }
26 
27         public IActionResult OnGet()
28         {
29             return Page();
30         }
31 
32         [BindProperty]
33         public Sneaker Sneaker { get; set; } = default!;
34 
35         public async Task<IActionResult> OnPostAsync()
36         {
37             if (Sneaker == null)
38             {
39                 return Page();
40             }
41 
42             Stopwatch stopwatch = new Stopwatch();
43             stopwatch.Start();
44 
45            imageFile1 = Request.Form.Files.GetFile("picture1");
46            imageFile2 = Request.Form.Files.GetFile("picture2");
```

```
47
48              MemoryStream dataStream = new MemoryStream();
49              await imageFile1.CopyToAsync(dataStream);
50              Sneaker.Picture1 = dataStream.ToArray();
51              dataStream = new MemoryStream();
52              await imageFile2.CopyToAsync(dataStream);
53              Sneaker.Picture2 = dataStream.ToArray();
54
55              _context.Sneakers.Add(Sneaker);
56              await _context.SaveChangesAsync();
57
58              stopwatch.Stop();
59             _logger.LogInformation("Sneaker Create Time: {0}",
                   stopwatch.ElapsedMilliseconds);
60
61              return RedirectToPage("/Index");
62          }
63      }
64  }
```

U prikazanom kodu je istaknuto kreiranje `ILogger<TCategoryName>` objekta koristeći injektiranje ovisnosti. Kreiran je `logger` objekt tipa `CreateModel`, taj naziv kategorije predstavlja niz znakova koji su zatim povezani uz svaki zapis. Nakon zaustavljanja štoperice poziva se naredba `LogInformation` koja formatira i zapisuje poruku informacijske razine ozbiljnosti. [26]

Kao što je vidljivo u kodu ne vrši se mjerenje izvršavanja same operacije prema bazi podataka već cijela aktivnost vezana uz tu operaciju što uključuje i dodatnu obradu podataka ili druge dijelove koda potrebne za ispravan rad aplikacije. Svaka poruka koja se zapisuje je napisana tako da opisuje događaj za koji je vrijeme mjereno, u ovom slučaju poruka glasi „*Sneaker Create Time:* „ kako bi se kasnije podaci mogli lakše grupirati i formatirati za daljnju analizu.

5.2. Testni podaci

Za testiranje aplikacije svaka baza je napunjena sa istim testnim podacima kako bi se izbjegle nepotrebne varijacije kod testiranja i mjerenja brzine. U svaku bazu podataka su dodane 22 tenisice gdje svaka tenisica sadrži dvije slike preuzete s interneta, zatim je za svaku tenisicu dodano nekoliko stavki inventara što ukupno rezultira sa 60 zapisa u inventaru. Nakon toga je uz administratora dodano još 9 običnih korisnika te su za svakog korisnika kreirane dvije narudžbe, jedna s jednom tenisicom i druga s dvije različite tenisice, za kraj je kod svakog korisnika ostavljen jedan artikl u košarici. Koristeći profil administratora naizmjenično su promijenjeni statusi svih kreiranih narudžbi.

5.3. Procedura mjerenja

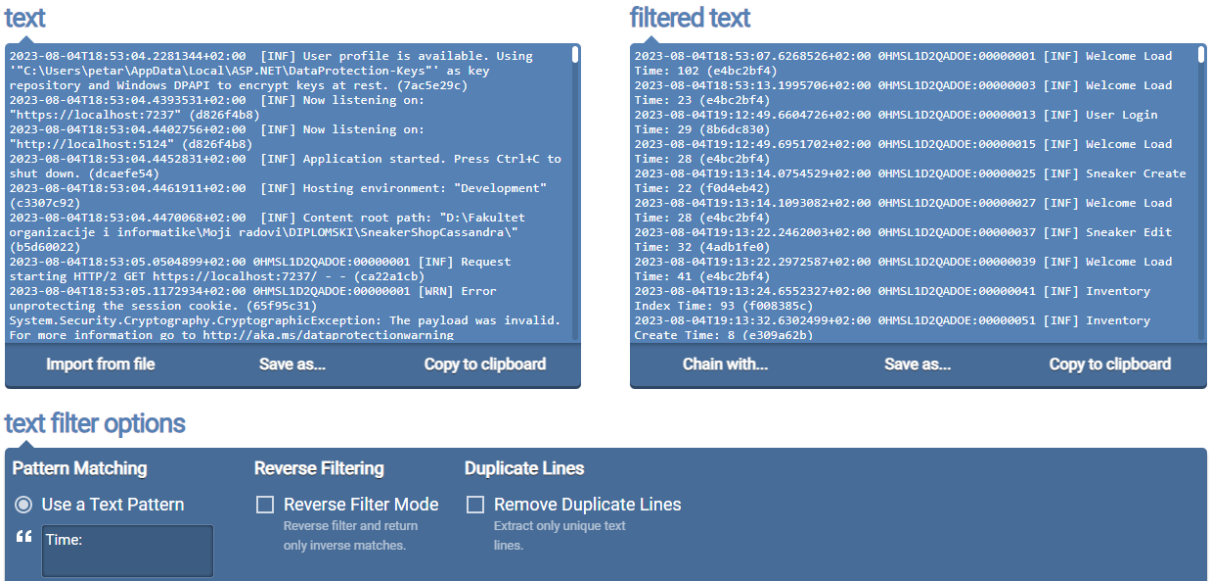
Kod samog procesa mjerenja brzine aplikacije kreirana je procedura koja prolazi kroz sve funkcionalnosti aplikacije te kroz sve dijelove koda koji mjere i zapisuju proteklo vrijeme. Procedura se za svaku bazu podataka izvršava 5 puta kako bi se prikupila zadovoljavajuća količina podataka za kasniju analizu. Koraci procedure su sljedeći:

- Filtriranje i sortiranje tenisica
- Prijava administratora
    - Dodavanje tenisice
        - Uređivanje tenisice
            - Dodavanje inventara
                - Uređivanje inventara
            - Brisanje inventara
    - Brisanje tenisice
    - Promjena statusa nekoliko narudžbi
    - Pregled svih korisnika
- Odjava administratora
- Registracija novog korisnika
    - Prijava novog korisnika
        - Dodavanje dvaju različitih tenisica u košaricu
        - Brisanje jedne tenisice iz košarice
        - Naručivanje sadržaja košarice
        - Pregled vlastitih narudžbi
        - Uređivanje profila
    - Brisanje profila

Kod koraka dodavanja tenisice kreira se novi zapis u bazi koji sadrži uz ostala ispunjena polja i dvije slike te tenisice koje su preuzete lokalno, budući da se ta ista tenisica u kasnijem koraku briše, kod svake iteracije navedene procedure se dodaje ista tenisica. Isto tako uvijek se kreira i briše isti korisnik zbog jednostavnosti obavljanja testiranja.

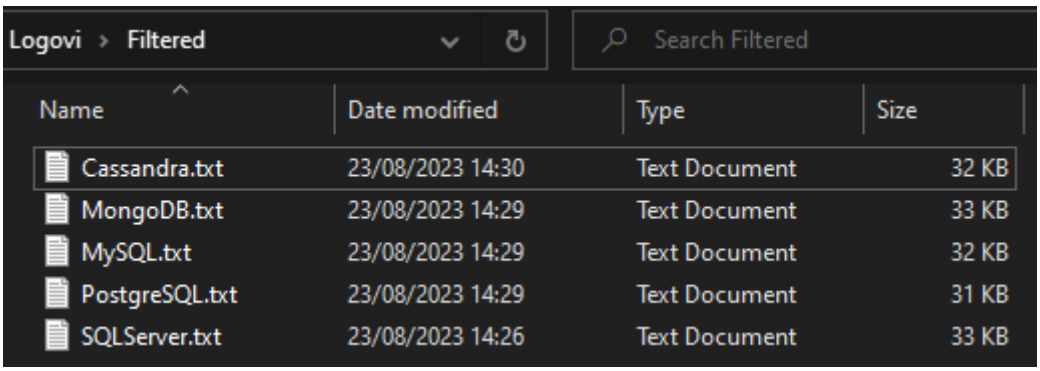
5.4. Obrada podataka

Prikupljeni podaci se nalaze u prethodno spomenutoj datoteci koja se generira prilikom pokretanja aplikacije. Uz potrebne zapise koji sadrže izmjerena vremena postoji i vrlo puno ostalih zapisa iste ili više razine ozbiljnosti pa je prvi korak pri obradi podataka filtriranje samo potrebnih podataka iz datoteke. Za filtriranje je korišten online alat dostupan na https://onlinetexttools.com/filter-text.



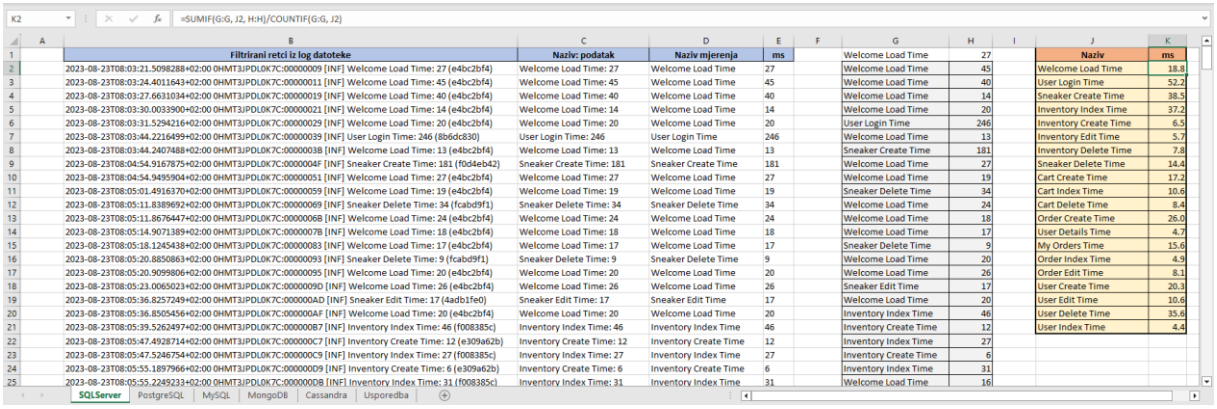
Slika 20: OnlineTextTools alat za filtriranje teksta (vlastita izrada)

Na slici iznad prikazano je sučelje alata za filtriranje, budući da datoteka dnevnika stavlja svaki zapis u novi red moguće je uz pomoć ovog alata upisati uzorak teksta te zatim alat izdvoji linije koje sadrže zadani uzorak. U ovom slučaju je iskorištena karakteristika poruka koje opisuju svako od izmjerenih vremena a to je da njihov opis završava sa tekstom „Time:“ pa je isti tekst korišten za filtriranje tih linija. Filtrirani tekst je kopiran i spremljen u zasebnu datoteku za arhiviranje i daljnju obradu, ovo su datoteke s filtriranim podacima:



Slika 20: Tekstualne datoteke s filtriranim zapisima dnevnika (vlastita izrada)

Sljedeće je kreirana Excel datoteka sa jednim listom za svaku bazu podatka. Kopiran je sadržaj datoteka sa slike 20 te je iz svake linije dnevnika izdvojena poruka koja opisuje mjerenu aktivnost te sama izmjerena vrijednost.



Filtrirani retci iz log datoteke	Naziv: podatak	Naziv mjerenja	ms
2023-08-23T08:03:21.5098288+02:00 0HMT3JPDL0K7C:00000009 [INF] Welcome Load Time: 27 (e4bc2bf4)	Welcome Load Time: 27	Welcome Load Time	27
2023-08-23T08:03:24.4011643+02:00 0HMT3JPDL0K7C:00000011 [INF] Welcome Load Time: 45 (e4bc2bf4)	Welcome Load Time: 45	Welcome Load Time	45
2023-08-23T08:03:27.6631034+02:00 0HMT3JPDL0K7C:00000019 [INF] Welcome Load Time: 40 (e4bc2bf4)	Welcome Load Time: 40	Welcome Load Time	40
2023-08-23T08:03:30.0033900+02:00 0HMT3JPDL0K7C:00000021 [INF] Welcome Load Time: 14 (e4bc2bf4)	Welcome Load Time: 14	Welcome Load Time	14
2023-08-23T08:03:31.5294216+02:00 0HMT3JPDL0K7C:00000029 [INF] Welcome Load Time: 20 (e4bc2bf4)	Welcome Load Time: 20	Welcome Load Time	20
2023-08-23T08:03:44.2216499+02:00 0HMT3JPDL0K7C:00000039 [INF] User Login Time: 246 (8b6dc830)	User Login Time: 246	User Login Time	246
2023-08-23T08:03:44.2407488+02:00 0HMT3JPDL0K7C:0000003B [INF] Welcome Load Time: 13 (e4bc2bf4)	Welcome Load Time: 13	Welcome Load Time	13
2023-08-23T08:04:54.9167875+02:00 0HMT3JPDL0K7C:0000004F [INF] Sneaker Create Time: 181 (f0d4eb42)	Sneaker Create Time: 181	Sneaker Create Time	181
2023-08-23T08:04:54.9495904+02:00 0HMT3JPDL0K7C:00000051 [INF] Welcome Load Time: 27 (e4bc2bf4)	Welcome Load Time: 27	Welcome Load Time	27
2023-08-23T08:05:01.4916370+02:00 0HMT3JPDL0K7C:00000059 [INF] Welcome Load Time: 19 (e4bc2bf4)	Welcome Load Time: 19	Welcome Load Time	19
2023-08-23T08:05:11.8389692+02:00 0HMT3JPDL0K7C:00000069 [INF] Sneaker Delete Time: 34 (fcabd9f1)	Sneaker Delete Time: 34	Sneaker Delete Time	34
2023-08-23T08:05:11.8676447+02:00 0HMT3JPDL0K7C:00000068 [INF] Welcome Load Time: 24 (e4bc2bf4)	Welcome Load Time: 24	Welcome Load Time	24
2023-08-23T08:05:14.9071389+02:00 0HMT3JPDL0K7C:0000007B [INF] Welcome Load Time: 18 (e4bc2bf4)	Welcome Load Time: 18	Welcome Load Time	18
2023-08-23T08:05:18.1245438+02:00 0HMT3JPDL0K7C:00000083 [INF] Welcome Load Time: 17 (e4bc2bf4)	Welcome Load Time: 17	Welcome Load Time	17
2023-08-23T08:05:20.8850863+02:00 0HMT3JPDL0K7C:00000093 [INF] Sneaker Delete Time: 9 (fcabd9f1)	Sneaker Delete Time: 9	Sneaker Delete Time	9
2023-08-23T08:05:20.9099806+02:00 0HMT3JPDL0K7C:00000095 [INF] Welcome Load Time: 20 (e4bc2bf4)	Welcome Load Time: 20	Welcome Load Time	20
2023-08-23T08:05:23.0065023+02:00 0HMT3JPDL0K7C:0000009D [INF] Welcome Load Time: 26 (e4bc2bf4)	Welcome Load Time: 26	Welcome Load Time	26
2023-08-23T08:05:36.8257249+02:00 0HMT3JPDL0K7C:000000AD [INF] Sneaker Edit Time: 17 (4adb1fe0)	Sneaker Edit Time: 17	Sneaker Edit Time	17
2023-08-23T08:05:36.8505456+02:00 0HMT3JPDL0K7C:000000AF [INF] Welcome Load Time: 20 (e4bc2bf4)	Welcome Load Time: 20	Welcome Load Time	20
2023-08-23T08:05:39.5262497+02:00 0HMT3JPDL0K7C:000000B7 [INF] Inventory Index Time: 46 (f008385c)	Inventory Index Time: 46	Inventory Index Time	46
2023-08-23T08:05:47.4928714+02:00 0HMT3JPDL0K7C:000000C7 [INF] Inventory Create Time: 12 (e309a62b)	Inventory Create Time: 12	Inventory Create Time	12
2023-08-23T08:05:47.5246754+02:00 0HMT3JPDL0K7C:000000C9 [INF] Inventory Index Time: 27 (f008385c)	Inventory Index Time: 27	Inventory Index Time	27
2023-08-23T08:05:55.1897966+02:00 0HMT3JPDL0K7C:000000D9 [INF] Inventory Create Time: 6 (e309a62b)	Inventory Create Time: 6	Inventory Create Time	6
2023-08-23T08:05:55.2249233+02:00 0HMT3JPDL0K7C:000000DB [INF] Inventory Index Time: 31 (f008385c)	Inventory Index Time: 31	Inventory Index Time	31

Welcome Load Time	27	
Welcome Load Time	45	
Welcome Load Time	40	
Welcome Load Time	14	
Welcome Load Time	20	
User Login Time	246	
Welcome Load Time	13	
Sneaker Create Time	181	
Welcome Load Time	27	
Welcome Load Time	19	
Sneaker Delete Time	34	
Welcome Load Time	24	
Welcome Load Time	18	
Welcome Load Time	17	
Sneaker Delete Time	9	
Welcome Load Time	20	
Welcome Load Time	26	
Sneaker Edit Time	17	
Welcome Load Time	20	
Inventory Index Time	46	
Inventory Create Time	12	
Inventory Index Time	27	
Inventory Create Time	6	
Inventory Index Time	31	
Welcome Load Time	16	

Naziv	ms
Welcome Load Time	18.8
User Login Time	52.2
Sneaker Create Time	38.5
Inventory Index Time	37.2
Inventory Create Time	6.5
Inventory Edit Time	5.7
Inventory Delete Time	7.8
Sneaker Delete Time	14.4
Cart Create Time	17.2
Cart Index Time	10.6
Cart Delete Time	8.4
Order Create Time	26.0
User Details Time	4.7
My Orders Time	15.6
Order Index Time	4.9
Order Edit Time	8.1
User Create Time	20.3
User Edit Time	10.6
User Delete Time	35.6
User Index Time	4.4

Slika 21: Excel list obrade podataka za SQLServer

Na slici iznad vidljiva je obrada podataka iz datoteke do razine da se za sa svaku mjerenu aktivnost dobije prosjek izmjerenih vrijednosti. Stupci G i H su kopirani stupci D i E zbog lakše daljnje obrade, također vrijednosti u stupcu H su pretvorene u brojčane vrijednosti kako bi se u konačnoj tablici s desne strane mogao izračunati prosjek pomoću sljedeće formule:

=SUMIF(G:G, J2, H:H)/COUNTIF(G:G, J2)

Ukratko ova formula grupira poruke aktivnosti te izračunava prosječno vrijeme za svaku poruku. Za kraj je dodan još jedan list gdje su spojeni svi obrađeni podaci kako bi se mogli koristiti za analizu u sljedećem poglavlju ovog rada.

6. Analiza rezultata

U ovom poglavlju će biti analizirani prikupljeni podaci o brzini rada pojedinih dijelova aplikacije. Kao što je već prije navedeno vremenska mjerenja se ne odnose na čisto izvršavanje upita prema bazi već na vrijeme potrebno da se izvrši dio koda vezan za mjerenu aktivnost, taj dio koda može sadržavati prvo dohvaćanje potrebnih podataka iz baze, pa zatim obradu podataka prikupljenih iz forme, formatiranje novog objekta i upita, izvršavanje novog upita te ažuriranje aplikacije s novim informacijama. Bitno je imati na umu što je zapravo mjereno kako bi se mogli pravilno analizirati podaci. Analiza je provedena tako da su podaci prikupljeni u prethodnom poglavlju spojeni u jednu finalnu tablicu koja sadrži izmjerena vremena za sve baze podataka, ovako izgleda ta tablica:

Primarna CRUD operacija	Naziv mjerenja	Vrijeme (ms)				
		SQLServer	PostgreSQL	MySQL	MongoDB	Cassandra
Čitanje	Učitavanje početne stranice	18.8	7.6	16.4	7.5	8.6
	Prijava korisnika	52.2	38.0	41.2	10.4	13.6
	Učitavanje inventara	37.2	14.9	22.3	17.9	36.5
	Učitavanje košarice	10.6	6.2	6.2	20.1	3.2
	Učitavanje svih narudžbi	4.9	3.5	4.4	9.5	21.6
	Učitavanje svih korisnika	4.4	3.8	4.4	1.4	1.3
	Učitavanje profila	4.7	5.5	4.1	1.5	1.1
	Učitavanje narudžbi korisnika	15.6	10.3	21.0	21.1	3.6
Kreiranje	Dodavanje tenisice	38.5	45.6	51.2	21.4	9.2
	Dodavanje inventara	6.5	4.5	8.2	5.1	6.3
	Dodavanje u košaricu	17.2	13.4	17.5	5.7	17.6
	Kreiranje narudžbe	26.0	29.4	27.8	24.6	42.4
	Registracija korisnika	20.3	10.8	12.8	7.4	7.6
Ažuriranje	Ažuriranje inventara	5.7	3.3	5.1	1.8	8.9
	Mijenjanje statusa narudžbe	8.1	7.0	10.1	5.6	8.5
	Ažuriranje korisnika	10.6	9.0	10.0	7.6	7.0
Brisanje	Brisanje inventara	7.8	8.1	10.1	4.4	13.5
	Brisanje tenisice	14.4	7.0	10.0	5.0	12.4
	Brisanje iz košarice	8.4	7.3	9.0	4.5	12.4
	Brisanje korisnika	35.6	7.2	9.8	4.2	12.2

Tablica 1: Podaci o brzini rada aplikacije ovisno o bazi podataka (vlastita izrada)

Kao što je vidljivo na prikazanoj tablici mjerenja su poredana drugačijim redoslijedom nego što je vidljivo na slici 21. Razlog tome je što sam odlučio dodatno grupirati mjerenja po

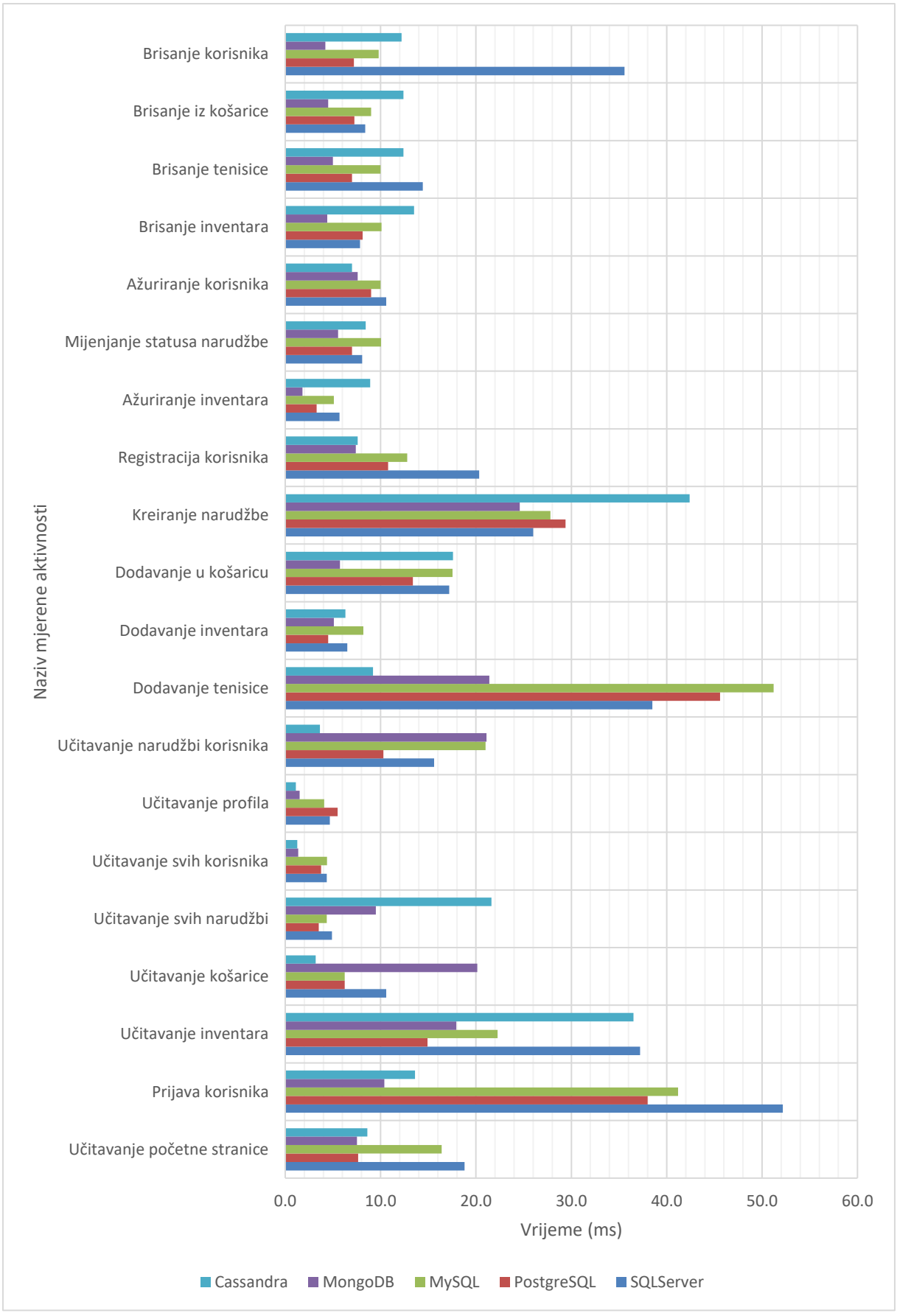
njihovoj primarnoj CRUD operaciji. Time se misli da na primjer aktivnost dodavanja u košaricu nema nužno samo operaciju dodavanja novog retka u tablicu košarice već se izvršavaju operacije čitanja podataka iz tablica tenisice i inventara kako bi se radile provjere dostupne količine ali primarna operacija i svrha te aktivnosti je dodati/kreirati novi zapis u košarici. Koristeći to grupiranje kreirana je još jedna tablica iz koje će biti izvučeni grafički prikazi pri analizi:

Primarna CRUD operacija	Vrijeme (ms)				
	SQLServer	PostgreSQL	MySQL	MongoDB	Cassandra
Čitanje	18.5	11.2	15.0	11.2	11.2
Kreiranje	21.7	20.7	23.5	12.8	16.6
Ažuriranje	8.1	6.4	8.4	5.0	8.1
Brisanje	16.6	7.4	9.7	4.5	12.6

Tablica 2: Prosječne brzine rada aplikacije ovisno primarnoj CRUD operaciji i bazi podataka (vlastita izrada)

6.1. Usporedba brzine aplikacije ovisno o bazi podataka

Koristeći podatke iz tablice 1 kreiran je grafički prikaz za lakšu usporedbu podataka. U ovom poglavlju će biti analizirani podaci svake od mjerenih aktivnosti te opisan razlog kod nekih ekstremnijih rezultata.



Slika 22: Grafički prikaz brzina aktivnosti aplikacije ovisno o bazi podatka (vlastita izrada)

Sa slike 22 možemo izvaditi puno zaključaka o utjecaju odabira baze podataka na samu brzinu programske aplikacije. Za početak možemo vidjeti da kod aktivnosti brisanja korisnika najbrže vrijeme ima aplikacija koja koristi MongoDB bazu podataka, nešto sporija je PostgreSQL verzija, dok su MySQL i Cassandra otprilike duplo sporije u usporedbi s MongoDB bazom. Daleko najsporija baza podataka u ovom testiranju je SQLServer, razlog tome je što je dodan kod za brisanje korisničkih narudžbi kod brisanja korisnika budući da SQLServer u ovom slučaju nema kaskadno brisanje.

Brisanje artikla iz košarice ima vrlo slične rezultate prethodnoj aktivnosti osim SQLServer-a, ponovno MongoDB ima najmanje izmjereno vrijeme izvršavanja aktivnosti dok Cassandra rezultira skoro tri puta duljim vremenom. SQLServer se kod ovog brisanja nalazi u sredini.

Kod aktivnosti brisanja tenisice ponovno možemo vidjeti dominaciju MongoDB baze po brzini, druga po redu najbrža baza podataka je PostgreSQL dok su ostale otprilike duplo sporije.

Brisanje inventara ima slične rezultate brisanju iz košarice, MongoDB ima najbrža vremena a druga po redu baza podataka je čak SQLServer, kod ove aktivnosti najsporijom bazom podataka se pokazala Cassandra.

Aktivnost ažuriranja korisnika ima vrlo slične rezultate mjerenja brzine kod svih baza podataka ali i dalje je vidljivo da najkraće vrijeme imaju Cassandra i MongoDB dok su ostale baze podataka nešto malo sporije.

Mijenjanje statusa narudžbe, također aktivnost ažuriranja baze podataka, prikazuje da je ponovno MongoDB najbrža baza, malo sporija je PostgreSQL pa SQLServer dok najgore rezultate imaju Cassandra i MySQL.

Ažuriranje inventara ima velike razlike u brzini od baze do baze podataka, daleko najbrže vrijeme izvršavanja ima ažuriranje u MongoDB bazi dok je Cassandra znatno sporija od ostalih baza.

Aktivnost registracije korisnika je najbrža kod Cassandre i MongoDB baze podataka, nešto sporiji su rezultati za MySQL i PostgreSQL dok je MySQL daleko najsporija baza u ovom mjerenju.

Kod kreiranja narudžbi se izvodi više različitih upita na bazu te nešto kompleksniji blokovi koda što je vidljivo zbog ukupnog povećanja vremena izvršavanja aktivnosti kod svih baza podataka. Ističe se najsporije vrijeme kod Cassandra baze, dok su sve ostale baze podataka brže za otprilike 40%.

Dodavanje tenisice u košaricu prikazuje ponovno značajni utjecaj MongoDB baze na brzinu izvršavanja, vidljivo je da je MongoDB duplo brži od PostgreSQL te tri puta brži od ostalih baza podataka.

Kod aktivnosti dodavanja inventara odabir baze podataka nije imao znatan utjecaj na brzinu, vidljivo je da su kroz sve baze podataka brzine vrlo male te je ukupna razlika svega nekoliko milisekundi.

Dodavanje tenisice, aktivnost kod koje se u bazu podataka spremaju slike tenisice u obliku bajtova, prikazuje zanimljive rezultate i dominaciju ne relacijskih baza podataka. Cassandra ima daleko najbrže vrijeme, slijedi ju duplo sporiji MongoDB dok su ostale, relacijske baze, prosječno dva puta sporije od MongoDB-a

Učitavanje narudžbi korisnika je aktivnost koja znatno varira u implementaciji ovisno o bazi podataka, Cassandra je u ovom mjerenju ponovno daleko najbrža, možemo zaključiti da je to zbog korištenja korisnički definiranih tipova i ugniježđenog spremanja podataka usprkos duplikaciji samih podataka. Kod Cassandre je u ovom slučaju vrlo lagano pristupiti podacima koji su potrebni za prikaz narudžbi korisnika dok se kod ostalih baza izvršavaju dodatni upiti i spajanja tablica kako bi se dobili isti rezultati.

Aktivnosti učitavanja profila i učitavanja svih korisnika imaju gotovo jednake rezultate, vidljivo je da su Cassandra i MongoDB baze znatno utjecale na brzinu rada aplikacije što je rezultiralo tri do pet puta boljim izmjerenim vremenom od ostalih baza podataka.

Učitavanje svih narudžbi, aktivnost dostupna administratoru, prikazuje neočekivano spore rezultate za Cassandra bazu podataka, jedan o potencijalnih razloga je to što je inventar tablica, odnosno kolekcija s najviše zapisa, a budući da je u ovom radu Cassandra postavljena samo na jedan lokalni čvor nije iskorišten potpun potencijal baze. Najbrža baza podataka za ovu aktivnost se pokazala PostgreSQL, blizu iza nje su MySQL i SQLServer dok je MongoDB duplo sporiji od njih.

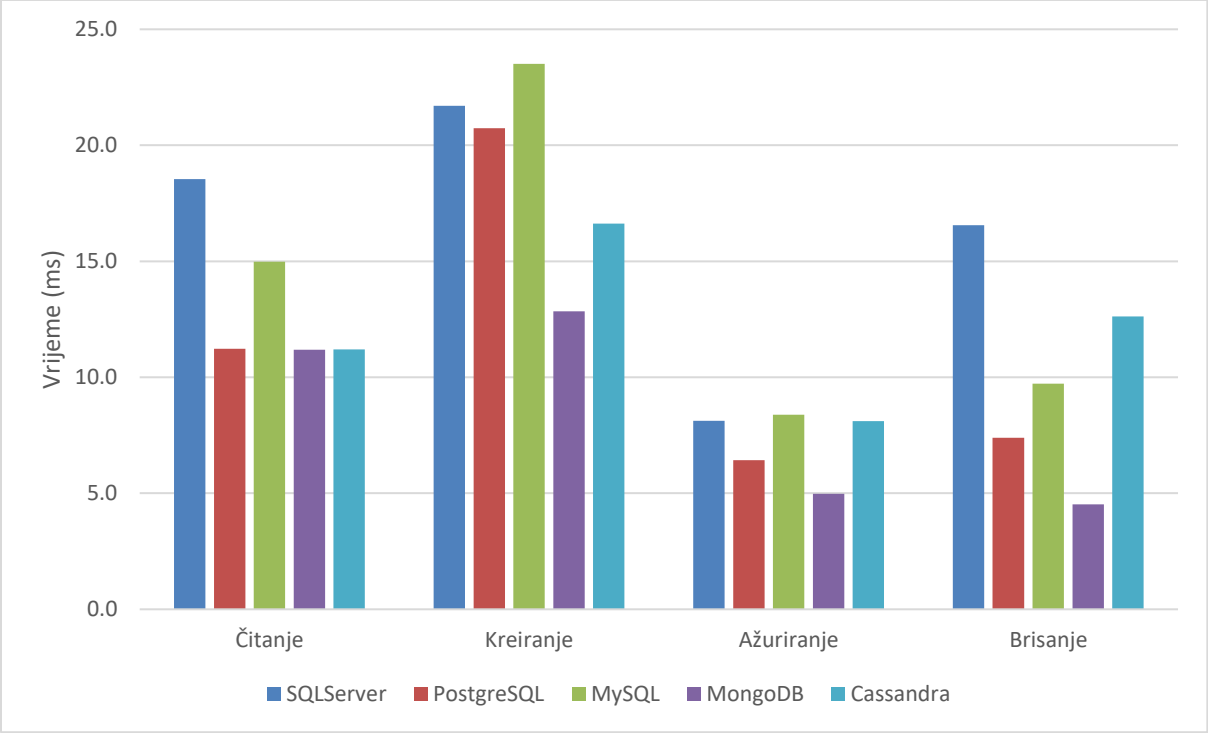
Kod učitavanja košarice su vidljivi slični rezultati kao i kod učitavanja narudžbi korisnika, Cassandra je ponovno najbrža zbog ugniježđenog spremanja podataka dok se kod ostalih baza podatak spajaju tablice za kreiranje istih rezultata.

Učitavanje inventara ima zanimljive rezultate, najbržom bazom podataka se pokazao PostgreSQL, kojeg u brzini slijede MongoDB i MySQL, Cassandra i SQL Server su otprilike duplo sporiji kod ove aktivnosti.

Prijava korisnika je najbrže izvršena kod Cassandra i MongoDB baze podataka dok su ostale baze nekoliko puta sporije.

Posljednja izmjerena aktivnost, učitavanje početne stranice, prikazuje da su Cassandra, MongoDB i PostgreSQL podjednako najbrže dok su MySQL i SQLServer duplo sporije.

Analizom izmjerenih brzina rada aplikacije za odabrane aktivnosti vidjeli smo detaljan utjecaj odabira baze podataka no za lakše predočavanje utjecaja na pojedinu vrstu aktivnosti koristit će se podaci iz tablice 2 u kojoj su do sada analizirane vrijednosti dodatno grupirane prema primarnoj CRUD operaciji svake od aktivnosti. Grafički prikaz dobiven iz tih podataka izgleda ovako:



Slika 23: Grafički prikaz brzine aplikacije ovisno o vrsti aktivnosti i bazi podataka (vlastita izrada)

Iz slike 23 možemo analizirani prednosti i mane, točnije pozitivne i negativne utjecaje svake od baza podataka na brzinu rada programske aplikacije. SQL Server se pokazao kao najsporija baza podataka kod aktivnosti gdje je primarna CRUD operacija brisanje i čitanje, te druga najsporija baza podataka u ostalim kategorija što u prosijeku čini ovu baza podatak najsporijom od svih testiranih.

PostgreSQL je zajedno sa MongoDB i Cassandrom najbrža baza podataka u aktivnostima gdje je primarna CRUD operacija čitanje iz baze. Druga je po redu kod aktivnosti ažuriranja i brisanja te srednja kod aktivnosti gdje je primarna operacija kreiranje zapisa u bazi.

MySQL ima najsporije rezultate vezane uz operacije kreiranja i ažuriranja te se nalazi u sredini kod aktivnosti čitanja i brisanja.

MongoDB ima vrlo impresivne rezultate jer kao što je vidljivo na slici 23, ova baza podataka ima najbrža izmjerena vremena u svim kategorija. Znatno je brža kod aktivnosti gdje su primarne CRUD operacije kreiranje, ažuriranje i brisanje dok kod aktivnosti s primarnom operacijom čitanja podataka iz baze dijeli prvo mjesto sa PostgreSQL i Cassandra bazom.

Posljednja baza podataka, Cassandra, ima najbrže vrijeme samo u jednog kategoriji a to je kod aktivnosti gdje je primarna CRUD operacija čitanje. Druga je po redu kod kreiranja zapisa u bazi ali je i druga najsporija što se tiče ažuriranja i brisanja.

Gledajući ukupno prosječno vrijeme svih izmjerenih aktivnosti MongoDB je najbrža baza podataka sa prosječnim vremenom od 9.3 ms. Slijedeći MongoDB nekoliko milisekundi kasnije su PostgreSQL i Cassandra, dvije baze podataka vrlo blizu u prosjeku, sa vremenima od 12.1 ms za PostgreSQL i 12.4 za Cassandru. Nešto sporije baze podataka što je vidljivo i po dosadašnjoj analizi rezultata su MySQL sa prosječnim vremenom od 15.1 ms i SQLServer sa 17.4 ms.

6.2. Usporedba SQL i NoSQL baza podataka

Kod odabira baze podataka za neki projekt ili aplikaciju jedno od glavnih pitanja i odluka je odabir između dvije vrste baza podataka: relacijske, SQL baze i ne-relacijske NoSQL baze.

Strukturirani upitni jezik (eng. *Structured Query Language – SQL*) se koristi u sustavima za upravljanje relacijskim baza od 1970-ih. Svrha baza podataka koje koriste SQL jezik je smanjiti dupliciranje podataka te njihova vrijednost leži u upravljanju strukturiranim podacima koji sadrže relacije odnosno čiji su podaci međusobno povezani. Također SQL baze podataka kao što smo već vidjeli u ovom radu, spremaju podatke u tablice te koriste striktno definiranu shemu. NoSQL ili ne samo SQL (eng. *Not only SQL*) baze podataka omogućavaju rad s različitim strukturama podataka te nisu striktno vezane za zapisivanje u tablice. Dinamična shema ne relacijskih baza podataka rezultira manjom potrebnom za prethodno planiranje i organizaciju podataka te olakšava kasnije promijene u bazi. [27]

U ovom poglavlju će biti uspoređene SQL i NoSQL baze podataka korištene u ovom radu, od toga su SQLServer, MySQL i PostgreSQL predstavnici SQL, relacijskih baza podataka dok MongoDB i Cassandra predstavljaju neke od popularnijih ne-relacijskih, NoSQL baza. Kako bi mogli analizirati utjecaj spomenutih kategorija na brzinu rada aplikacije, podaci iz prethodno prikazanih tablica su oblikovani tako da su izračunate prosječne vrijednosti

podataka koji su dobiveni mjerenjem određenih aktivnosti u aplikaciji ovisno o kategoriji u koju baza podataka pripada. Stoga su dobivene sljedeće tablice:

Primarna CRUD operacija	Naziv mjerenja	Vrijeme (ms)	
		SQL	NoSQL
Čitanje	Učitavanje početne stranice	14.3	8.1
	Prijava korisnika	43.8	12.0
	Učitavanje inventara	24.8	27.2
	Učitavanje košarice	7.7	11.7
	Učitavanje svih narudžbi	4.3	15.6
	Učitavanje svih korisnika	4.2	1.3
	Učitavanje profila	4.7	1.3
	Učitavanje narudžbi korisnika	15.6	12.4
Kreiranje	Dodavanje tenisice	45.1	15.3
	Dodavanje inventara	6.4	5.7
	Dodavanje u košaricu	16.0	11.7
	Kreiranje narudžbe	27.7	33.5
	Registracija korisnika	14.6	7.5
Ažuriranje	Ažuriranje inventara	4.7	5.4
	Mijenjanje statusa narudžbe	8.4	7.0
	Ažuriranje korisnika	9.9	7.3
Brisanje	Brisanje inventara	8.7	9.0
	Brisanje tenisice	10.5	8.7
	Brisanje iz košarice	8.2	8.5
	Brisanje korisnika	17.5	8.2

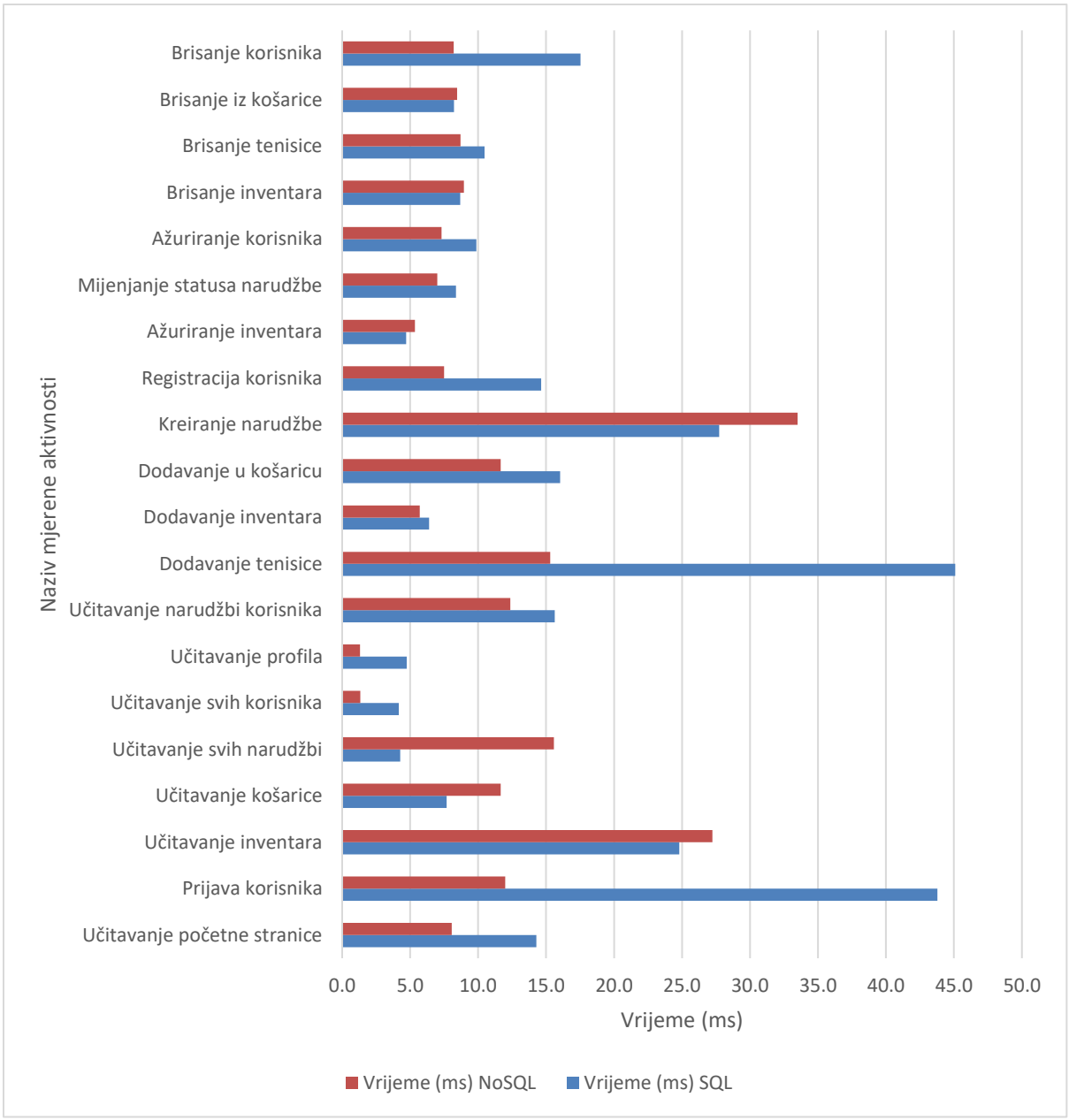
Tablica 3: Podaci o brzini rada aplikacije ovisno o vrsti baze podataka (vlastita izrada)

Sukladno tablici 2 i korištenjem iste logike, iz tablice 3 je kreirana dodatna tablica koja dodatno pojednostavljuje i grupira podatke kako bi se dobili finalni rezultati ovog rada, tablica izgleda ovako:

Primarna CRUD operacija	Vrijeme (ms)	
	SQL	NoSQL
Čitanje	14.9	11.2
Kreiranje	22.0	14.7
Ažuriranje	7.6	6.6
Brisanje	11.2	8.6

Tablica 4: Prosječne brzine rada aplikacije ovisno primarnoj CRUD operaciji i vrsti baze podataka (vlastita izrada)

U nastavku će biti prikazani grafovi vezani uz tablice kako bi lakše analizirali podatke.

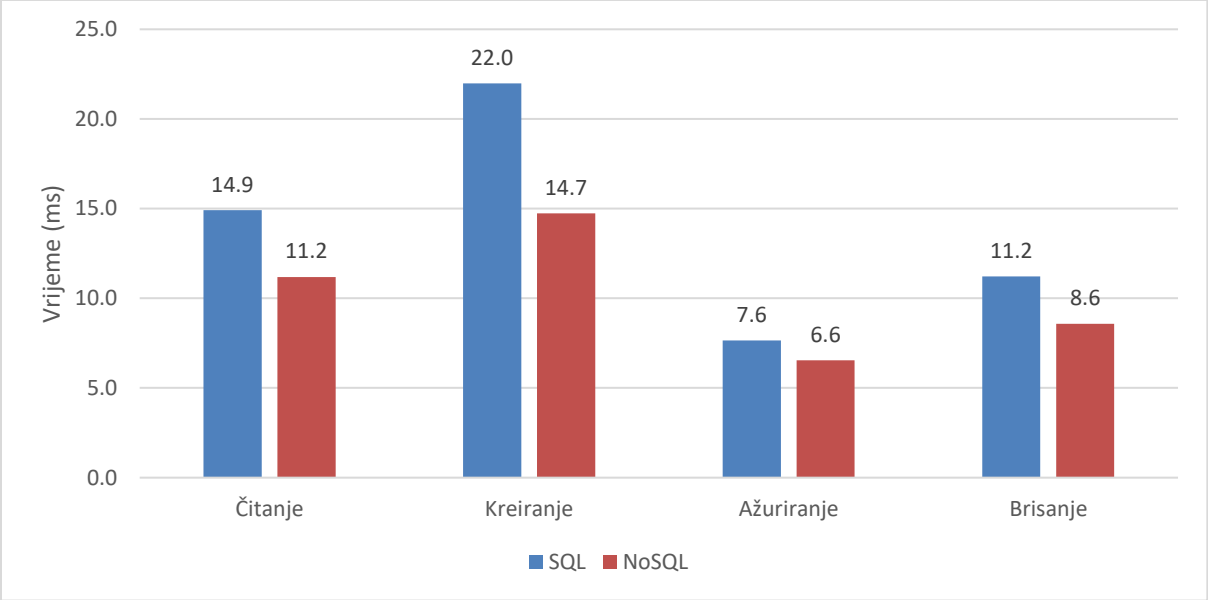


Slika 24: Grafički prikaz brzina aktivnosti aplikacije ovisno o vrsti baze podatka (vlastita izrada)

Iz prikazanog grafa vidljivo je da korištenje SQL baze podataka rezultira bržim izvršavanjem aktivnosti samo u nekoliko slučajeva, zanemarivši aktivnosti gdje su rezultati gotovo jednaki možemo zaključiti da je SQL baza podataka brža kod aktivnosti gdje se čitaju kompleksniji podaci odnosno gdje se radi više spajanja ili upita kako bi se dobili potrebni podaci. To je vidljivo kod aktivnosti učitavanja svih narudžbi, učitavanja košarice te učitavanja inventara gdje veliku ulogu igra i implementacija baze podataka i model koji je korišten, budući da se u ovom radu koristi model koji više odgovara relacijskim bazama te se kod ne-relacijskih simulira korištenje tablica ili se koristi ugnježđivanje podataka kako bi se što manje mijenjala

programska logika za bolju usporedbu. Smatram da bi dodatnim optimiziranjem aplikacije i kompletnim redizajnom modela podataka za korištenje u ne-relacijskim bazama podataka izmjerena vremena izvršavanja bila još manja te bi NoSQL baze podataka bile brže u gotovo svakoj mjerenoj aktivnosti. Usprkos tome iz grafa možemo zaključiti da kod ove usporedbe korištenje NoSQL baza podataka rezultira bržim izvršavanjem programske aplikacije.

Za posljednju analizu iz tablice 4 je kreiran graf prikazan u nastavku kako bi se vidjelo kako vrsta baze podatak utječe na brzinu izvršavanja aktivnosti grupiranih po primarnoj CRUD operaciji koju aktivnost obavlja prema bazi podataka.



Slika 25: Grafički prikaz brzine aplikacije ovisno o vrsti aktivnosti i vrsti baze podataka (vlastita izrada)

Iz prikazanog grafa možemo zaključiti da korištenje NoSQL baze podataka rezultira bržim radom aplikacije. Najveća razlika u ovoj usporedbi je vidljiva kod aktivnosti gdje je primarna CRUD operacija kreiranje zapisa u bazi podataka kod koje korištenje SQL baze rezultira skoro 50% duljim vremenom izvršavanja. Odabir vrste baze podataka ima najmanji utjecaj kod aktivnosti ažuriranja gdje je razlika u prosjeku izmjerenih brzina samo jedna milisekunda.

7. Zaključak

Ovim radom sam obradio temu utjecaja odabira baze podataka na brzinu rada aplikacije. Upoznao sam se sa ASP.NET Core okvirom za izradu web aplikacija te uspješno kreirao aplikaciju koju sam zatim spojio na pet različitih baza podataka kako bi se što bolje usporedio njihov utjecaj na brzinu rada.

Budući da je ovaj rad više praktično i analitički orijentiran bilo je vrlo zanimljivo učiti i raditi s tehnologijama koje sam odabrao. Sviđa mi se sloboda koju sam imao kod odabira alata koji je korišten, vrste aplikacije i baza podataka. Zbog toga sam i odabrao ASP.NET Core okvir koji se koristi u meni već dobro poznatom Visual Studio-u te je cijela aplikacija napisana korištenjem C# jezika. Smatram da mi je odabir ovih tehnologija pomogao pri održavanju konzistentnosti između implementacija različitih baza podataka jer okvir nudi strukturiran i standardiziran pristup kreiranju same web aplikacije i načina povezivanja bilo koje baze podataka. Što se tiče samih baza podataka odabrao sam SQLServer, MySQL i PostgreSQL jer su među najpopularnijim bazama na svijetu dok su Cassandra i MongoDB najpoznatije baze za svoju kategoriju odnosno vrstu. Za svaku korištenu bazu podataka sam prvo ukratko opisao karakteristike te zatim prikazao instalaciju potrebnih programa, postupak kreiranja same baze i prikladnih tablica ili kolekcija, zatim kako se baza povezuje sa programskom aplikacijom i kako se kreiraju i izvršavaju CRUD operacije. Nakon upoznavanja s aplikacijom i bazama opisano je mjerenje same brzine aplikacije, kako je implementirano u kodu, koji su podaci bili korišteni te koja je bila procedura za prikupljanje podataka. Za kraj su podaci obrađeni i analizirani te sam zaključio da je MongoDB najbrža baza podataka kod ovakve primjene. Svaka baza podataka ima svoje prednosti i mane te idealni scenarij u kojem bi korištenje pojedine baze imalo najviše smisla i gdje bi se maksimalno iskoristio njezin potencijal, zbog toga je potrebno dobro analizirati i ocijeniti karakteristike i potrebe aplikacije.

Ovim radom sam definitivno stekao uvid u implementaciju i upravljanje odabranim bazama podataka te sam sretan što sam odlučio koristiti ASP.NET Core okvir za izradu aplikacije jer su mi se svidjele funkcionalnosti koje pruža te smatram da sam stekao korisno znanje za buduće programiranje. Iznenađen sam rezultatima jer sam očekivao da će SQL Server biti jedna od bržih baza podataka, jednim dijelom zbog reputacije a drugim dijelom zbog odlične integracije u Visual Studio-u. Također bi dodao da se korištene NoSQL baze podataka mogu još ubrzati ako bi se više mijenjala logika aplikacije i model podataka što čini dobivene rezultate još impresivnijima jer su se i ovako implementirane pokazale kao najbrže. Smatram da se definitivno isplati učiti raditi sa ne-relacijskim bazama podataka i natjerati se razmišljati izvan tablice jer kao što je prikazano ovim radom to rezultira boljim performansama aplikacije.

Popis literature

[1] Microsoft, „Visual Studio 2022“, (bez dat.) [Na internetu]. Dostupno: https://visualstudio.microsoft.com/ [pristupano 19.08.2023.]

[2] Microsoft, „IntelliSense in Visual Studio“, ožujak 2023. [Na internetu]. Dostupno: https://learn.microsoft.com/en-us/visualstudio/ide/using-intellisense?view=vs-2022 [pristupano 19.08.2023]

[3] D. Roth, R. Anderson, S. Luttin, „Overview of ASP.NET Core“, studeni 2022. [Na internetu]. Dostupno: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0 [pristupano 19.08.2023.]

[4] Microsoft, „Choose an ASP.NET Core web UI“, travanj 2023. [Na internetu]. Dostupno: https://learn.microsoft.com/en-us/aspnet/core/tutorials/choose-web-ui?view=aspnetcore-7.0 [pristupano 19.08.2023.]

[5] R. Anderson, J. Smith, „Part 5, work with a database in an ASP.NET Core MVC app“, svibanj 2023. [Na internetu]. Dostupno: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/working-with-sql?view=aspnetcore-7.0&tabs=visual-studio [pristupano 19.08.2023.]

[6] D. Lim, „What’s an Example of Good E-Commerce Database Design?“, studeni 2020. [Na internetu]. Dostupno: https://fabric.inc/blog/commerce/ecommerce-database-design-example [pristupano 19.08.2023.]

[7] G. Mortier, „Ecommerce Database Design: ER Diagram for Online Shopping“, travanj 2023. [Na internetu]. Dostupno: https://vertabelo.com/blog/er-diagram-for-online-shop/ [pristupano 19.08.2023.]

[8] R. Anderson, D. Brock, K. Larkin, „Introduction to Razor Pages in ASP.NET Core“, ožujak 2023. [Na internetu]. Dostupno: https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-7.0&tabs=visual-studio [pristupano 19.08.2023]

[9] M. Varga, „Baze podataka“, srpanj 2022. [Na internetu]. Dostupno: https://books.google.hr/books?id=UQPoDwAAQBAJ&printsec=frontcover&hl=hr#v=onepage&q&f=false [pristupano 19.08.2023.]

[10] DB-Engines, „DB-Engines Ranking“, kolovoz 2023. [Na internetu]. Dostupno: https://db-engines.com/en/ranking [pristupano 19.08.2023.]

[11] M. Ilić, L. Kopanja, D. Zlatković, M. Trajković, D. Ćurguz, „MICROSOFT SQL SERVER AND ORACLE: COMPARATIVE PERFORMANCE ANALYSIS“, lipanj 2021. [Na internetu]. Dostupno: http://kmi.vtsns.edu.rs/KMI_2021/radovi/1-KMI_Informatika/KMI_informatika-1.5.pdf [pristupano 19.08.2023.]

[12] D. Damodaran, S. Salim, S. M. Vargese „Performance evaluation of MySQL and MongoDB databases“, travanj 2016. [Na internetu]. Dostupno: https://d1wqtxts1xzle7.cloudfront.net/59431690/PERFORMANCE_EVALUATION_OF_MYSQL_AND20190528-97716-g7l9vr-libre.pdf?1559105800=&response-content-disposition=inline%3B+filename%3DPERFORMANCE_EVALUATION_OF_MYSQL_AND_MONG.pdf [pristupano 19.08.2023.]

[13] MongoDB, „Introduction to MongoDB“, (bez dat.) [Na internetu]. Dostupno: https://www.mongodb.com/docs/manual/introduction/ [pristupano 19.08.2023.]

[14] P. Khandelwal, S. Addie, „Create a web API with ASP.NET Core and MongoDB“, travanj 2023. [Na internetu]. Dostupno: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mongo-app?view=aspnetcore-7.0&tabs=visual-studio [pristupano 19.08.2023.]

[15] „Cassandra Basics“, (bez dat.) [Na internetu]. Dostupno: https://cassandra.apache.org/_/cassandra-basics.html [pristupano 19.08.2023.]

[16] E. Hewitt, „Cassandra: The Definitive Guide“, studeni 2010. [Na internetu]. Dostupno: https://books.google.hr/books?hl=hr&lr=&id=MKGSbCbEdg0C&oi=fnd&pg=PR7&dq=cassandra+database&ots=XrYx7yF95A&sig=9_PnXqOnGapVqCifPHK5SwSn9nk&redir_esc=y#v=onepage&q&f=false [pristupano 19.08.2023.]

[17] V. Kaplarevic, „How to Install Cassandra on Windows 10“, (bez dat.) [Na internetu]. Dostupno: https://phoenixnap.com/kb/install-cassandra-on-windows [pristupano 19.08.2023.]

[18] DataStax, „Data replication“, (bez dat.) [Na internetu]. Dostupno: https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/architecture/archDataDistributeReplication.html [pristupano 19.08.2023.]

[19] Apache Cassandra, „Data Types“, (bez dat.) [Na internetu]. Dostupno: https://cassandra.apache.org/doc/latest/cassandra/cql/types.html [pristupano 19.08.2023.]

[20] MySQL, „What is MySQL?“, (bez dat.) [Na internetu]. Dostupno: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html [pristupano 19.08.2023.]

[21] Oracle, „What is MySQL?“, (bez dat.) [Na internetu]. Dostupno: https://www.oracle.com/mysql/what-is-mysql/ [pristupano 19.08.2023.]

[22] PostgreSQL, „About“, (bez dat.) [Na internetu]. Dostupno: https://www.postgresql.org/about/ [pristupano 19.08.2023.]

[23] J. C. Worsley, J. D. Drake, „Practical PostgreSQL“, siječanj 2002. [Na internetu]. Dostupno: https://books.google.hr/books?hl=hr&lr=&id=fI1lAgAAQBAJ&oi=fnd&pg=PR4&dq=pos

tgresql+database&ots=a00G94O_z1&sig=LBWUdr07IxKuLx18ShaUtUkB8QE&redir_esc=y#v=onepage&q=postgresql%20database&f=false [pristupano 19.08.2023.]

[24] S. Juba, A. Volkov, „Learning PostgreSQL 11“, siječanj 2019. [Na internetu]. Dostupno: https://books.google.hr/books?hl=hr&lr=&id=ZtOGDwAAQBAJ&oi=fnd&pg=PP1&dq=postgresql+database&ots=GF1IJc3eNE&sig=rOXLRI0dYP3x7R3Xdx-Adboj7_A&redir_esc=y#v=onepage&q=postgresql%20database&f=false [pristupano 19.08.2023.]

[25] Microsoft, „Stopwatch Class“, (bez dat.) [Na internetu]. Dostupno: https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.stopwatch?view=net-7.0 [pristupano 19.08.2023.]

[26] K. Larkin, J. Gutsch, R. Anderson, „Logging in .NET Core and ASP.NET Core“, ožujak 2023. [Na internetu]. Dostupno: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/logging/?view=aspnetcore-7.0 [pristupano 19.08.2023.]

[27] B. Anderson, B. Nicholson, „SQL vs. NoSQL Databases: What’s the Difference?“, lipanj 2022. [Na internetu]. Dostupno: https://www.ibm.com/blog/sql-vs-nosql/ [pristupano 19.08.2023.]

Popis slika

Popis tablica