Primjer baze podataka u sustavu za upravljanje bazama podataka Redis

Baričević, Vedran

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Vedran Baričević

Primjer baze podataka u sustavu za upravljanje bazama podataka Redis

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Vedran Baričević

Matični broj: 44015/17–R

Studij: Primjena informacijske tehnologije u poslovanju

Primjer baze podataka u sustavu za upravljanje bazama podataka Redis

ZAVRŠNI RAD

Mentor/Mentorica:

Izv. prof. dr. sc. Schatten Markus

Varaždin, rujan 2021.

Vedran Baričević

Izjava o izvornosti

Izjavljujem da je moj završni/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/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Završni rad se sastoji od dva dijela: teorijski i praktični. U teorijskom dijelu obrađuju se nerelacijske baze podataka općenito. Prolazi se kroz povijest nerelacijskih baza podataka te kako su se one razvile u moderne baze podataka. Uspoređuju se nerelacijske s relacijskim bazama podataka po svojstvima kao što su struktura, shema, opseg radnog opterećenja i skaliranje. Opisuju se prednosti i nedostaci nerelacijskih baza podataka te donosi zaključak na temelju njih. U drugom teorijskom dijelu obrađuje se Redis, što je tema ovog rada. Redis je opisan kroz sve njegove tipove podataka te kroz najpopularnije slučajeve njegovog korištenja.

U praktičnom dijelu opisuje se aplikacija koju je autor napravio za ovaj rad. Aplikacija u pitanju je internetska trgovina igara napravljena u .NET CORE programskom jeziku. Aplikacija koristi MySQL za primarnu bazu podataka te Redis za sekundarnu bazu podataka. Svrha Redisa je da pamti koliko se puta korisnik neuspješno prijavio, da pamti korisničke postavke na stranici te da pamti stavke koje se nalaze u korisničkoj košarici.

Kroz rad se želi dokazati da su suvremene nerelacijske baze podataka vrijedne za razmotriti i koristiti pri početku projekta, što je i zaključak rada.

Ključne riječi: Redis, baze podataka, NoSQL, nerelacijske baze podataka, programiranje, sustav za upravljanje bazama podataka;

Sadržaj

1. Uvod

Tema ovog rada je opisati sustav za upravljanje bazama podataka Redis te napraviti praktični primjer aplikacije koja koristi Redis kao jednu od baza podataka. Autor je odabrao temu jer je već bio upoznat s relacijskim bazama podataka, ali nije gotovo ništa znao o nerelacijskim baza podataka te je htio proširiti svoje znanje o njima. Istina je da su nerelacijske baze podataka popularne, ali nisu ni približno popularne kao relacijske baze podataka iz razloga kojih ćemo se dotaći i u samom radu. Kroz rad ćemo pokazati da nerelacijske baze podataka mogu biti jednako korisne kao i relacijske baze podataka.

Temu ćemo kroz rad obraditi kroz tri cjeline: NoSQL baze podataka, Redis i praktični primjer. U prvoj cjelini opisati ćemo što su NoSQL baze podataka, njihovu povijest, usporediti ih s SQL bazama podataka te navesti njihove prednosti i nedostatke. U drugoj cjelini opisati ćemo Redis, njegovu sintaksu, tipove podataka i slučajeve korištenja. U trećoj cjelini ćemo kroz praktični primjer pokazati neke primjene Redisa kao sekundarne baze podataka u .NET CORE *Web* aplikaciji. Aplikacija je *online* trgovina koja koristi primarno koristi Redis za pamćenje korisnikove košarice pri kupnji te još neke primjene koje ćemo opisati u radu.

Cilj ovog rada je dokazati da je Redis moćan sustav za upravljanje bazama podataka, visokih performansi, je jednostavan za korištenje te ima specifične slučajeve korištenja ali je odličan izbor za te slučajeve korištenja.

2. NoSQL baze podataka

2.1. Povijest NoSQL baza podataka

Prve baze podataka koje su se pojavile bile su relacijske baze podataka, izumljene od strane Edgar F. Codda 1970. godine. Relacijske baze podataka sadrže podatke raspoređene u različite redove i stupce povezujući ključeve s određenim redcima (Foote, 2018.). Gotovi svi takvi sustavi koriste SQL (engl. *structured query language*) te su iznimno složeni. Takvi sustavi su vrlo kontrolirani i imaju ograničenu mogućnost prevođenja složenih, nestrukturiranih podataka. Ipak, SQL sustavi se i dalje opsežno koriste i vrlo su korisni za veliki broj slučajeva korištenja za organizacije bilo kakvih veličina (Foote, 2018.).

Sredinom 1990-ih godina internet je postao iznimno popularan te relacijske baze podataka jednostavno nisu mogle držati korak s povećanim protokom informacija koje su zahtijevali korisnici, zajedno s većom raznolikošću tipova podataka koju je donio taj rast (Foote, 2018.). Ova evolucija je uzrokovala rast nerelacijskih baza podataka, koje se često nazivaju još i NoSQL baze podataka. Takve baze podataka mogu s lakoćom prevesti nestrukturirane podatke i izbjeći krutost SQL-a, pri čemu zamjenjuju organiziranost pohrane s većom fleksibilnošću (Foote, 2018.).

Akronim NoSQL je 1998. godine prvi put upotrijebio Carlo Strozzi pri imenovanju svoje relacijske baze podataka otvorenog koda koja nije koristila SQL (Foote, 2018.). Akronim se opet pojavio 2009. godine kada su ga Eric Evans i Johan Oskarsson iskoristili da opišu nerelacijsku bazu podataka. Danas se termin NoSQL koristi za opisivanje nečega što ne koristi SQL sustave ili ne koristi samo SQL (Foote, 2018.).

CAP teorem, koji je također poznat i kao Brewerov teorem (nazvat tako po osobi koja ga je razvila, Ericu Breweru), je vrlo važan dio NoSQL baza podataka. Prema CAP teoremu, distribuirano skladište podataka ne može istodobno nuditi više od dva od tri utvrđena jamstva (Foote, 2018.). Prema Footeu [1], u toj teoriji, tri utvrđena jamstva koja se ne mogu istodobno ispuniti su sljedeća:

- Dosljednost (engl. *consistency*) – podaci unutar baze podataka uvijek ostaju konzistentni, čak i nakon izvođenja operacija nad njima. Na primjer, nakon ažuriranja sustava svi bi klijenti trebali vidjeti jednake podatke

- Dostupnost (engl. *availability*) – sustav cijelo vrijeme radi bez zastoja

- Particijska tolerancija (engl. *partition tolerance*) – čak i u slučajevima kada komunikacija između poslužitelja više nije pouzdana, sustav će nastaviti s funkcioniranjem. To se događa iz razloga što se poslužitelji mogu podijeliti u više grupa koje ne mogu više komunicirati

Pošto se istodobno ne mogu dobiti sva tri, postoje tri moguće kombinacije dizajna distribuiranog spremišta podataka, prema IBM-u [4] :

- CA (konzistentnost, dosljednost) – Podaci su konzistentni između svih čvorova te sve dok su povezani na mrežu bi trebali omogućiti svim klijentima pisanje i čitanje čvorova

- CP (konzistentnost, particijska tolerancija) – Podaci su konzistentni između čvorova te postaju nedostupni kada se bilo koji čvor odspoji s mreže kako bi se održavala particijska tolerancija

- AP (dostupnost, particijska tolerancija) – Mogućnost čvorova da ostanu na mreži čak i kada ne mogu komunicirati međusobno. Nakon što je particioniranje završeno, čvorovi će pokušati sinkronizirati podatke, ali bez garancije da će podaci biti jednaki

2002. godine MIT-jevci Nancy Lynch i Seth Gilbert objavili su službeni dokaz Brewerova teorema te time dokazali da je teorem istinit.

1983. godine Theo Härder i Andreas Reuter izmislili su popularan model konzistencije ACID (Foote, 2018.). Prema Scholesu [5], akronim ACID predstavlja:

- atomarnost (engl. *atomicity*) – Svi dijelovi transakcije se tretiraju kao jedna radnja. Pri izvršavanja transakcije, izvrše se svi njeni dijelovi ili se odbacuje cijela transakcija

- konzistentnost (engl. *consistency*) – Nijedna transakcija ne smije uzrokovati nedozvoljeno stanje podataka. U prijevodu, svaka transakcija mora pratiti pravila definirana od strane baze podataka.

- izolacija (engl. *isolation*) – Osigurava da je rezultat svake transakcije stanje sustava koje bi se dobilo kao da su se transakcije izvršile jedna nakon druge

- izdržljivost (engl. *durability*) – Jednom kada je transakcija izvršena, ona se neće poništiti čak i ako postoje sukobi s drugim operacijama

Prednost koju ACID pruža je sigurno okruženje za procesiranje podataka, gdje su podaci stabilni, konzistentni te se mogu spremati u više memorijskih lokacija. ACID je posebno koristan kod grafičkih NoSQL baza podataka koji ga koriste da osiguraju da su podaci sigurno i konzistentno spremljeni (Foote, 2018.).

Drugi popularni model je BASE, koji je 2008. godine postao alternativa ACID modelu. Prema Poreu [2], akronim BASE predstavlja:

- Jednostavno dostupan (engl. *basically available*) – Sustav jamči da će biti dostupan čak i u slučaju kvara

- meko stanje (engl. *soft state*) – Mogućnost promjene stanja podataka bez interakcije aplikacije zbog konačne dosljednosti

- konačna dosljednost (engl. *eventual consistency*) – Mogućnost sustava da ne garantira dosljednost na nivou transakcije, ali nakon ulaza (engl. *input*) aplikacije će u konačnici doći do stanja dosljednosti

BASE model ima slabu dosljednost. U zamjenu za to, on nudi horizontalno skaliranje, toleranciju grešaka i visoku dostupnost. Dostupnost za skaliranje je važna značajka spremišta podataka koje koriste BASE model. BASE model najčešće koriste baze podataka dokumenta, stupčaste i ključ – vrijednost baze podataka (Foote, 2018.).

NoSQL baze podataka su nerelacijske, bez sheme i horizontalno skalirane. Takve baze podataka su vrlo elastične i omogućavaju brze promjene, ali zato većinom podupiru samo jednostavne sustave (Foote, 2018.). Prema Footeu [1], četiri najpopularnije vrste NoSQL baza podataka su:

- Ključ – vrijednost (engl. *key – value*)

- Spremište dokumenata

- Širokog stupca

- Grafička

Ključ – vrijednost baze podataka su dizajnirane za pohranu, dohvaćanje i upravljanje asocijativnim nizovima. To znači da svaki zapis u ključ – vrijednost bazi podataka ima ključ i vrijednosti koje su asocirane s tim ključem. Prednost takvih baza podataka je to što su vrlo jednostavne i lagane za implementirat. S druge strane, jednom kada je zapis upisan u bazu podataka, teško ga je djelomično ažurirati. Također, ovakve baze podataka nisu namijenjene za složene sustave ili da se koriste kao primarna baza podataka. Ključ – vrijednost baze podataka se često, na primjer, koriste za natjecateljske ljestvice ili za košarice u internetskim trgovinama. Primjeri ključ – vrijednost baza podataka su Redis, Oracle NoSQL, Amazon DynamoDB i MemcacheDB [6].

Spremišta dokumenata su sustavi napravljeni za pohranu, dohvaćanje i upravljanje informacija orijentiranih na dokumente (Foote, 2018.). Dokumenti u pitanju su zapravo skupine drugih ključ – vrijednost skupina te su pohranjeni u formati kao što su JSON [7]. Razlika između ključ – vrijednost baza podataka i spremišta dokumenata je ta što spremišta dokumenata dopuštaju da se ugniježđene vrijednosti asociraju s pojedinim ključem. Također, spremišta podataka spremaju sve informacije o nekom predmetu kao jednu instancu u bazi podataka te koriste unutarnju strukturu dokumenta za identifikaciju. Primjer spremišta dokumenata je MongoDB [7].

Baze podataka širokog stupca koriste tablice, redove i stupce kao i relacijske baze podataka. Razlika je u tome što se imena i formati stupaca mogu mijenjati iz reda u red unutar iste tablice (Foote, 2018.). U tablicama su prisutni ključevi ali mogu biti asocirani s više stupaca. Bazu podataka širokog stupca možemo interpretirati kao dvodimenzionalnu ključ – vrijednost bazu podataka. Primjeri baze podataka širokog stupca su Cassandra i HBase [8].

Grafičke baze podataka su dizajnirane za podatke čije veze su prikazane grafom koji sadrži elemente slične entitetima. Entiteti su povezani s drugim entitetima pomoću “rubova“ (engl. *edge*). Svaki entitet je opisan pomoću jedinstveni identifikator te niza rubova koja odlaze od njega ili dolaze od drugih entiteta. Također, svaki rub je opisan pomoću jedinstvenog identifikatora, svojeg početnog i/ili krajnjeg čvora te ostalih svojstava (Foote, 2018.). Podaci koji se pohranjuju u ovakve baze podataka su na primjer prometne veze, karte cesta ili mrežne topologije. Primjeri ovakvih baza podataka su Neo4J, OrientDB i Virtuoso [9].

2.2. Usporedba s SQL bazama podataka

S početkom gotovo bilo kojeg projekta treba odlučiti koju vrstu baze podataka koristiti. Kod te odluke uvijek je prvo pitanje “hoće li se koristiti relacijska ili nerelacijska baza podataka?“. Kako se u tom trenutku odlučiti za jednu od njih? Između njih postoji veliki broj razlika kao što je struktura, shema, skaliranje. U ovom dijelu proći ćemo kroz sve bitne razlike i odrediti za koje su slučajeve korištenja najzgodnije određene značajke.

2.2.1. Struktura

Struktura je način na koji su elementi uređeni te koje su veze između elemenata. Relacijska baza podataka koristi tablice u kojoj je jedan red jednak jednoj instanci entiteta u tablici, dok stupci prikazuju atribute entiteta. Svaki entitet posjeduje primarni ključ koji se koristi za jednoznačno identificiranje jedne instance entiteta. S druge strane, nerelacijske baze podataka koriste ključ – vrijednost parove koji omogućuju da se nekoliko povezanih stavki spremi u jedan red u tablici (Foote, 2018.). Također, treba napomenuti da jedan red u relacijskoj bazi podataka i jedan red u nerelacijskoj bazi podataka nije isto. Uzmimo za primjer jednog korisnika (Foote, 2018.). U nerelacijskoj bazi podataka svi podaci o nekom korisniku (ime, prezime, mjesto, broj mobitela, email itd.) bi bili spremljeni u jednom redu. S druge strane, u relacijskoj bazi podataka taj isti korisnik ne bi nužno imao sve svoje podatke raspoređene u jednom redu u tablici, nego preko više tablica. U relacijskoj bazi podataka može se dogoditi da korisnik ima atribut “mjesto“ koji je vanjski ključ na tablicu “mjesto“ koje ima atribute kao što su poštanski broj, pripadajuća županija i slično. Naravno, to ne dolazi bez svojih nedostataka. Nerelacijske baze podataka ne mogu imati veze između ključ - vrijednost parova kao što relacijske baze podataka mogu (Foote, 2018.). Struktura relacijskih baza podataka je napravljena u vrijeme kada su podaci bili uglavnom strukturirani te imali dobro definirane veze. Danas podaci nastoje biti puno složeniji i nestrukturirani (fotografije, video, mailovi i slično), za koje su nerelacijske baze podataka puno pogodnije. Relacijske baze podataka bi onda bile pogodnije za aplikacije koje koriste jednostavne, strukturirane podatke.

2.2.2. Shema

Velika prednost nerelacijskih baza podataka je to što imaju dinamičnu shemu. To joj dozvoljava da se ona mijenja bez ikakvih posljedica na trenutno stanje baze podataka. Također, to znači da se prije kreiranja same baze ne treba paziti na njenu shemu (Talha, 2021.). S druge strane, prije kreiranja relacijske baze podataka potrebno je prvo isplanirati i

definirati njenu shemu jer bez toga nije uopće moguće dodavati zapise u bazu podataka (Berga, Franco, 2021.). Također, poslije definiranja sheme teško je napraviti ikakve promjene jer ih već definirana shema neće dozvoliti. Mijenjanje same sheme relacijske baze podataka može biti vrlo skupo, dugotrajno te često dolazi uz zastoje ili prekide usluge. Iz tog razloga možemo zaključiti da su nerelacijske baze podataka pogodnije za projekte koji nemaju jasno definirane zahtjeve ili za projekte za koje znamo da će imati zahtjeve koji su podložni promjenama. Relacijske baze podataka bi onda bile pogodnije na primjer za jednostavne *web* aplikacije za koje imamo dobro definirane zahtjeve (Talha, 2021.).

2.2.3. Opseg radnog opterećenja

Već smo spomenuli da nerelacijske baze podataka koriste particijsku toleranciju. To znači da su odlične za rukovanje velikim količinama podataka te imaju impresivne performanse pri vraćanju podataka iz upita. To znači da, ako govorimo o terabajtima podataka, nerelacijske baze podataka će biti pogodnije pri rukovanje takvih količina podataka (Talha, 2021.). Dok relacijske baze podataka mogu u sebi sadržavati terabajte podataka, ako pokušamo preko upita dohvatiti bilo koju količinu podataka blizu tog broja, performanse pri obavljanju takvih poslova će biti nezadovoljavajuće (Talha, 2021.). Dakle, možemo zaključiti da su nerelacijske baze podataka bolje za aplikacije koje rukuju s velikim količinama podataka. Relacijske baze podataka se mogu koristiti za aplikacije koje sadrže veliku količinu podataka, ali ne bi trebale sadržavati upite koje mogu vratiti velike količine podataka.

2.2.4. Skaliranje

Skaliranje baze podataka je sposobnost proširenja kapaciteta sustava baze podataka kako bi se podržale veće količine, veći zahtjevi ili pohranilo više podataka bez žrtvovanja performansi. Prema Chinedu, skaliranje je moguće postići na dva različita načina:

- Vertikalno skaliranje
- Horizontalno skaliranje

Vertikalno skaliranje je opcija za skaliranje relacijskih baza podataka. Pošto je potreban samo jedan poslužitelj (engl. *server*) za posluživanje relacijske baze podataka i nema potrebe za više poslužitelja, skaliranje relacijske baze podataka se radi tako da se jednostavno kupi brži, veći, skuplji poslužitelj. Ovisno o veličini relacijske baze podataka ovo je tipično vrlo skupa opcija (Chinedu, 2021.).

Horizontalno skaliranje je način skaliranja nerelacijskih baza podataka. Jedna od prednosti horizontalnog skaliranja je takozvani *sharding*, što je metoda razdvajanja i jednog logičkog skupa podataka u više baza podataka (Chinedu, 2021.). To znači da jedna nerelacijska baza podataka može imati više “krhotina“ (engl. *shard*) koje predstavljaju jedan dio baze podataka, a mogu biti locirani bilo gdje u svijetu . Jednom “krhotinom“ upravlja jedan poslužitelj, a kada ona postane prevelika može se podijeliti na manje dijelove (Berga, Franco, 2021.). Pošto je kod ove metode skaliranja jedno ograničenje količina poslužitelja, rješenje za skaliranja nerelacijskih baza podataka je to da se jednostavno kupi više poslužitelja. Ovaj način skaliranja je jeftinije od vertikalnog skaliranja jer jer se kapacitet može dodati kupovanjem jeftinih poslužitelja.

2.3. Prednosti i nedostaci nerelacijskih baza podataka

Sada kada smo prošli kroz povijest nerelacijskih baza podataka i usporedili ih s relacijskim bazama podataka, dobili smo uvid u veliki broj njihovih svojstava koje mogu biti prednosti i nedostaci. U ovom odjeljku ćemo proći kroz najbitnije prednosti i nedostatke nerelacijskih baza podataka, počevši s prednostima.

2.3.1. Prednosti

- Visoke performanse – Kao što smo već rekli, nerelacijske baze podataka imaju puno bolje performanse od relacijskih baza podataka. Razloga tome je činjenica da relacijske baze podataka većinom koriste složene upite koji vraćaju podatke iz nekoliko tablica, dok kod nerelacijskih baza podataka svi podaci su ugniježđeni te zbog toga se i vraćaju puno brže. Neke vrste nerelacijskih baza podataka mogu obraditi i do 10.000 upita po sekundi, što je očito vrlo impresivno (Williams, 2021.).

- Fleksibilnost – Nerelacijske baze podataka su fleksibilne u mnogim aspektima. Za kreiranje nerelacijske baze podataka nije potrebno unaprijed definirana shema. Struktura podataka je vrlo fleksibilna te se može lako mijenjati u slučajevima kada su zahtjevi podložni promjenama (Williams, 2021.).

- Horizontalno skaliranje – Kao što smo već spomenuli nerelacijske baze podataka se skaliraju horizontalno, za razliku od relacijskih baza podataka koje se vertikalno skaliraju. Horizontalno skaliranje je puno jeftinije od vertikalnog skaliranja jer je samo

potrebno kupiti više poslužitelja te dodati na njega jednu “krhotinu“ baze podataka. S druge strane, za relacijske baze podataka je potrebno kupovati poslužitelje s boljim specifikacijama da bi se skalirale, što zna biti vrlo skupo (Williams, 2021.).

- Visoka dostupnost – Jedna od značajnijih prednosti nerelacijskih baza podataka je visoka dostupnost. No, što uopće znači visoka dostupnost? Najčešće to znači da baza mora raditi 99,9% vremena bez prekida. Veliki broj firmi danas ima usluge, kao na primjer internetske trgovine, koje zahtijevaju da im baza radi najmanje 99,9% vremena bez prekida. Zahtjevi su takvi jer potencijalni kupci mogu živjeti bilo gdje u svijetu te moraju moći pristupiti usluzi u bilo koje vrijeme. Iz tog razloga su nerelacijske baze podataka dobar izbor za sustave koji zahtijevaju visoku dostupnost.

- Specijalizirani modeli podataka – Već smo spomenuli neke nerelacijske modele podataka kao što su grafičke i za dokumente. No, postoji mnogo specijaliziranih modela podataka kao što su na primjer temporalne baze podataka. Takve baze podataka pohranjuju podatke koje se odnose na vremenske instance. Postoje hijerarhijske baze podataka koje su izvrsne za geoprostorne informacije koje se koriste u aplikacijama za geografsko označavanje (Williams, 2021.). Danas postoje mnogo takvih specifičnih slučajeva korištenja i modeli podataka nerelacijskih baza podataka mogu pokriti veliki dio njih.

Postoji još prednosti koje nismo spomenuli, ali one koje jesmo spomenuli su prilično značajne. U sljedećem dijelu proći ćemo kroz značajnije nedostatke nerelacijskih baza podataka te na kraju vidjeti jel mogu prevagnuti navedene prednosti.

2.3.2. Nedostaci

- Nedovoljno zreo – U usporedbi s relacijskim bazama podataka, NoSQL baze podataka nisu dovoljno zrele. Relacijske baze podataka su starije (izumljene 1970. godine), ali imaju i puno više novaca i truda investirano u njihov razvoj. Zbog toga su više funkcionalni i stabilniji kao sustavi (Williams, 2021.). Još jedna posljedica toga je činjenica da je lakše naći informacije i podršku za SQL, kao i stručnjake s kojima bi se mogli konzultirati za projekt ili zaposlenike koji bi mogli raditi na tom projektu. Isto tako relacijske baze podataka imaju više podrške koja je stalno dostupna (Williams, 2021.).

- Zahtjeva veći broj bazi podataka – Već smo rekli da NoSQL ima mnogo specifičnih slučajeva korištenja. Pri korištenju NoSQL-a u nekom projektu vrlo je moguće da će

biti potrebno koristiti više različitih baza i modela podataka u svrhu pokrivanja svih slučajeva korištenja koje projekt zahtjeva. S druge strane, SQL-ov model je vrlo generaliziran te se može koristiti za različite potrebe (Williams, 2021.).

- Konačna dosljednost – Kao što smo već rekli, nerelacijske baze podataka nude samo konačnu dosljednost. To znači da, iako baza podataka može imati neke nedosljednosti u bilo kojem trenutku, ona će s vremenom postati dosljedna kada sustav prekine s ažuriranjem. Na kraju će svi čvorovi primiti najnovija dosljedna ažuriranja (Harrison, 2011.). Iako ovo nije toliki nedostatak jer se pogreške naposljetku isprave, ipak je nedostatak i treba na njega tako i gledati.

2.3.3. Rezime

Sada kada smo prošli kroz sve prednosti i nedostatke, možemo ih sve zbrojiti te donijeti zaključak. Prednosti nerelacijskih baza podataka su: visoke performanse, fleksibilnost, horizontalno skaliranje, visoka dostupnost i specijalizirani modeli podataka. Nedostaci nerelacijskih baza podataka su: nedovoljna zrelost, sklonost zahtijevanju većeg broja bazi podataka i konačna dosljednost.

Prednosti	Nedostatci
Visoke performanse	Nedovoljna zrelost
Fleksibilnost	Zahtjeva veći broj baza podataka
Horizontalno skaliranje	Konačna dosljednost
Visoka dostupnost	
Specijalizirani modeli podataka	

Tablica 1. Usporedba prednosti i nedostataka

Prema tablici na slici broj 1 možemo vidjet da imamo pet prednost i tri nedostatka. Visoke performanse su velika prednost jer je to jedna od svojstva kojih želimo na bilo kojoj bazi podataka. Također, to je jedna od najbitnijih prednosti iz korisničke perspektive. Velika prednost su i specijalizirani modeli podataka koji osiguravaju da nerelacijske baze podataka

imaju mnogo slučajeva korištenja. S horizontalnim skaliranjem korištenje nerelacijskih baza podataka je relativno jeftino. S visokom dostupnosti nerelacijske baze podataka su dobar izbor za slučajeve korištenja koji zahtijevaju da je baza podataka gotovo cijelo vrijeme dostupna. Za fleksibilnost se može reći da je najmanja prednost koju smo naveli jer se prednosti fleksibilnosti sheme mogu izbjeći dobrim planiranjem.

Nedovoljna zrelost je definitivno najveći nedostatak nerelacijskih baza podataka te je vjerojatno najveći razlog zašto nerelacijske baze podataka nisu popularne kao relacijske baze podataka. Sklonost zahtijevanju većeg broja baza podataka je nedostatak ali je umanjen činjenicom da kupovanje novih poslužitelja nije skupo. Najmanji nedostatak nerelacijskih baza podataka je konačna dosljednost jer je to samo privremeni problem, ali je ipak problem. Kad se na kraju sve zbroji, prednosti nerelacijskih baza podataka nadmašuju njihove nedostatke.

3. Redis

Redis je sustav za upravljanje baza podataka otvorenog koda. Tipa je ključ – vrijednost te sprema te parove u radnu memoriju. Također se koristi za predmemoriranje (engl. *caching*) podataka i kao posrednik za poruke (engl. *message broker*). Razvijen je pomoću ANSI C jezika te je predviđen za rad na većini POSIX operacijskih sustava, ali Linux i OS X su sustavi na kojima Redis rade najoptimalnije [24]. Redis nema Windows distribuciju, ali postoji verzija Redisa koja je optimizirana za Windows. Ta verzija je 2.6, a trenutna stabilna verzija Redisa je 6.2. Također, Redis ima potporu za gotovo sve programske jezike, 54 jezika točnije. U ovom dijelu opisati ćemo sve bitne karakteristike Redisa, počevši sa sintaksom.

3.1. Sintaksa

Već smo rekli da je Redis baza podataka tipa ključ – vrijednost, ali važno je napomenuti da samo jedna od sedam Redisovih tipova podataka ima tipičnu ključ – vrijednost strukturu. Ti tipovi podataka su:

- Znakovni nizovi (engl. *string*)
- Liste
- Kriptografski sažeci (engl. *hash*)
- Skupovi
- Sortirani skupovi
- HyperLogLogs
- Tokovi

Prvo, započnimo s načinom kako pokrenuti Redis. U komandnu liniju na računalu na kojem je instaliran Redis upišemo *redis-cli*, što će pokrenuti i povezati se na lokalnog poslužitelja na portu 6379 (na uobičajenim postavkama). Nakon toga možemo upisat naredbu *PING* te ako smo pravilno spojeni na Redis poslužitelja, on će vratiti *PONG.* Tako znamo da Redis radi.

Sada možemo početi s popunjavanjem baze podataka.

3.1.1. Znakovni nizovi

Započnimo prvo sa znakovnim nizovima. Da bi postavili vrijednost tipa znakovni niz, koristimo naredbu *set*. Naredbi *set* moramo ponuditi još dva parametra, ključ i vrijednost. Prvi parametar je ključ, odnosno ime ključ, a drugi vrijednost koju želimo asocirati s tim ključem. Ako je naredba uspješno izvršena, Redis vraća odgovor *OK*. Važno je napomenuti da, ako

postoji već ključ s asociranom vrijednosti, naredba *set* će zamijeniti staru vrijednost s novom na tom ključu [17].

```
> set mykey somevalue
OK
> get mykey
"somevalue"
```

Slika 1. Set i get naredba

Dohvaćanje nekog znakovnog niza se radi pomoću naredbe *get*. Naredba *get* ima samo jedan parametar i to je naravno naziv ključa. Ako postoji zadani ključ, Redis vraća vrijednost koja je asocirana sa zadanim ključem [17]. Ako želimo provjeriti jel postoji neki ključ u bazi podataka, koristimo naredbu *exists*. Naredba ima jedan parametar, a to je naziv ključa kojeg tražimo te Redis vraća 1 ako on postoji ili 0 ako on ne postoji [17].

```
> set mykey hello
OK
> exists mykey
(integer) 1
> del mykey
(integer) 1
> exists mykey
(integer) 0
```

Slika 2. Exists naredba

Iako se tip podatka zove znakovni niz, njegova vrijednost također može biti broj. Nad brojevima možemo obavljati neke jednostavne operacije pomoću naredbi *incr, incrby, decr* i *decrby* [17]. Sve naredbe zahtijevaju naziv ključa za parametar, a vrijednost ključa mora biti broj. Naredbe *incr* i *decr* povećavaju ili smanjuju neku brojčanu vrijednost za 1, dok *incrby* i *decrby* povećavaju ili smanjuju neku brojčanu vrijednost za određeni iznos.

```
> set counter 100
OK
> incr counter
(integer) 101
> incr counter
(integer) 102
> incrby counter 50
(integer) 152
```

Slika 3. Incr i incrby naredba

Nadalje, moguće je napraviti da neki ključ ima određeno vrijeme za koje će biti prisutno u bazi podataka. To je moguće pomoću naredbe *expire* [17]. Također, možemo koristiti *set* te na kraju naredbe dodati *EX* što će postaviti to vrijeme pri postavljanju samog ključa. Naredba *expire* prima dva parametra, ključ i vrijeme. Vrijeme koje možemo postaviti pri korištenju tih naredbi može biti postavljeno u sekundama ili milisekundama [17].

```
> set key some-value
OK
> expire key 5
(integer) 1
> get key (immediately)
"some-value"
> get key (after some time)
(nil)
```

Slika 4. Expire naredba

Redis vraća 1 ako je zadano vrijeme postavljeno ili 0 ako ključ koji je dan kao parametar ne postoji. Ako pokušamo dohvatiti ključ za koji je vrijeme zadano naredbom *expire* isteklo, Redis vraća *(nil),* odnosno *null.* Ako želimo provjeriti koliko je još vremena preostalo nekom ključu, možemo koristiti naredbu *ttl*, što je skraćenica za vrijeme za živjeti (engl. *time to live*). Uz naredbu dajemo naziv ključa, a Redis vraća preostalo vrijeme koje je ostalo ključu [17].

```
> set key 100 ex 10
OK
> ttl key
(integer) 9
```

Slika 5. TTL naredba

Ako ipak želimo da neki ključ nema određeno vrijeme za život, koristimo naredbu *persist*. Jedini parametar je naziv ključa i Redis vraća 1 ako je preostale vrijeme maknuto ili 0 ako nije [17].

```
redis> SET mykey "Hello"
"OK"
redis> EXPIRE mykey 10
(integer) 1
redis> TTL mykey
(integer) 10
redis> PERSIST mykey
(integer) 1
redis> TTL mykey
(integer) -1
redis>
```

Slika 6. Persist naredba

Ako želimo izbrisati neki znakovni niz iz baze podataka, koristimo naredbu *getdel*. Jedini parametar naredbe *getdel* je naziv ključa.

```
redis> SET mykey "Hello"
"OK"
redis> GETDEL mykey
"Hello"
redis> GET mykey
(nil)
redis>
```

Slika 7. Getdel naredba

Ako se naredba uspješno izvrši, Redis vraća vrijednost ključa kojeg smo s naredbom izbrisali. Ako pokušamo opet dobiti vrijednost tog ključa s naredbom *get,* Redis vraća (nil) jer ključ više ne postoji.

3.1.2. Liste

Sljedeći tip podatka koji ćemo obraditi je lista. Liste se inicijaliziraju naredbom *lpush* ili *rpush*. One zahtijevaju naziv liste te naziv elementa koji se dodaje u listu pomoću naredbi. Te naredbe se također koriste za dodavanje elementa u listu ako ona već postoji, *lpush* dodaje element na početak liste te *rpush* na kraj [17].

```
> rpush mylist A
(integer) 1
> rpush mylist B
(integer) 2
> lpush mylist first
(integer) 3
```

Slika 8. Rpush i lpush naredba

Redis vraća broj elemenata u listi nakon dodavanja elementa ako se naredba uspješno izvršila. Vraćanje elemenata iz liste se vrši pomoću naredbe *lrange* koja prima naziv liste i dva indeksa kao parametre te vraća sve elemente čije su pozicije između ta dva indeksa. Ako je prvi indeks 0 i drugi -1, naredba vraća sve elemente liste [17].

```
> lrange mylist 0 -1
1) "first"
2) "A"
3) "B"
```

Slika 9. Lrange naredba

Uklanjanje elemenata iz liste može se postići s *rpop* s parametrom naziva liste, što miče element na zadnjem indeksu iz liste. Također, Redis vrati vrijednost koja je maknuta iz liste tako da se ova naredba može koristiti i za vraćanje vrijednosti [17].

```
> rpush mylist a b c
(integer) 3
> rpop mylist
"c"
> rpop mylist
"b"
> rpop mylist
"a"
```

Slika 10. Rpop naredba

3.1.3. Kriptografski sažeci

Redisovi kriptografski sažeci su vrlo slični znakovnim nizovima. Razlika je u tome što su kriptografski sažeci skupina ključ – vrijednost parova te nema određenog maksimalnog broja parova koje oni mogu sadržavati [17]. Gotovo sve naredbe koje se mogu izvršiti nad znakovnim nizovima mogu se izvršiti i na kriptografskim sažecima, pa tako imamo naredbe *hset, hget, hincrby* i *hexists* koje rade identično kao i pripadne naredbe koje rade na znakovnim nizovima. Također, imamo naredbu *hgetall* koja vraća sva polja i vrijednosti nekog kriptografskog sažetka.

```
redis> HSET myhash field1 "Hello"
(integer) 1
redis> HSET myhash field2 "World"
(integer) 1
redis> HGETALL myhash
1) "field1"
2) "Hello"
3) "field2"
4) "World"
redis>
```

Slika 11. Hset i Hgetall naredba

Moguće je i postaviti i dohvatiti više polja nekog kriptografskog sažetka s naredbama *hmset* i *hmget*. U slučaju naredbe *hmset* potrebno je navesti ime kriptografskog sažetka i bilo koji broj ključ – vrijednost parova [17].

```
redis> HMSET myhash field1 "Hello" field2 "World"
"OK"
redis> HGET myhash field1
"Hello"
redis> HGET myhash field2
"World"
redis>
```

Slika 12. Hmset i hget naredba

U slučaju naredbe *hmget* potrebno je samo navesti sve ključeve za koje želimo da se asocirana vrijednost vrati.

```
redis> HSET myhash field1 "Hello"
(integer) 1
redis> HSET myhash field2 "World"
(integer) 1
redis> HMGET myhash field1 field2 nofield
1) "Hello"
2) "World"
3) (nil)
redis>
```

Slika 13. Hmget naredba

Ako unesemo neki ključ koji nije prisutan u kriptografskom sažetku, onda Redis vraća (nil). Nad kriptografskim sažecima se također mogu vršiti naredbe koje bi se inače mogle izvršavati nad znakovnim nizovima, kao što su *hincr* i *hincrby* [17].

```
> hincrby user:1000 birthyear 10
(integer) 1987
> hincrby user:1000 birthyear 10
(integer) 1997
```

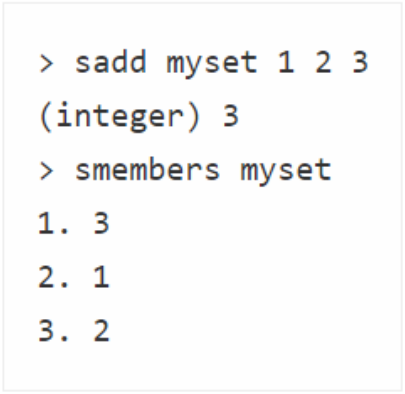
Slika 14. Hincrby naredba

Obje naredbe imaju istu funkciju, samo što se mogu jedino izvršavati nad kriptografskim sažecima.

3.1.4. Skupovi

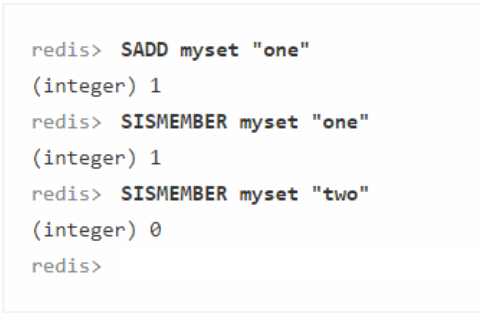
Sljedeći tip podatka je skup (engl. *set*). Skup je nesređena kolekcija znakovnih nizova. Kreiranje novog skupa i dodavanje novih članova u neki skup se vrši pomoću naredbe *sadd*.

Prvi parametar je naziv skupa, a nakon toga možemo specificirati bilo koji broj elemenata kojih želimo dodati u skup [17].



Slika 15. Sadd i smembers naredba

Ako se naredba uspješno izvršila, Redis vraća broj članova koji smo dodali u skup. Nakon što smo dodali nove članove, možemo provjeriti koji su svi članovi prisutni u skupu, a to radimo pomoću *smembers* naredbe. Jedini parametar koji je potreban je naziv skupa. Kao što vidimo na gornjoj slici, naredba *smembers* vraća sve članove skupa u nesređenom poretku . Također, možemo i provjeriti jel postoji određeni član nekog skupa pomoću naredbe *sismember* [17].



Slika 16. Sismember naredba

Prvi parametar naredbe je naziv skupa, a drugi je naziv člana. Ako je član prisutan u skupu, Redis vraća 1 te 0 nula ako nije. Pomoću naredbe *spop* možemo nasumično izbaciti i vratiti jednog člana iz skupa. Ako želimo nasumično izbaciti samo jednog člana iz skupa, naredbi dajemo samo jedan parametar, a to je naziv skupa [17].

```
redis> SADD myset "one"
(integer) 1
redis> SADD myset "two"
(integer) 1
redis> SADD myset "three"
(integer) 1
redis> SPOP myset
"two"
redis> SMEMBERS myset
1) "three"
2) "one"
```

Slika 17. Spop naredba

Ako se naredba uspješno izvrši, Redis će vratiti naziv člana koji se izbacio. Ako želimo nasumično izbaciti više članova iz skupa odjednom, osim imena skupa dajemo još i broj koji će odrediti koji broj članova želimo izbaciti iz skupa.

```
redis> SADD myset "four"
(integer) 1
redis> SADD myset "five"
(integer) 1
redis> SPOP myset 3
1) "five"
2) "four"
3) "three"
redis> SMEMBERS myset
1) "one"
```

Slika 18. Spop više članova

Ako želimo samo nasumično vratiti određeni broj članova iz skupa, koristimo naredbu *srandmember.* Osim toga, *srandmember* isto funkcionira kao *spop.* Još jedna razlika je u tome što naredbi možemo ponuditi negativni broj kao parametar, u kojem slučaju Redis može vratiti i članove koji nisu jedinstveni [17].

```
redis> SRANDMEMBER myset -5
1) "two"
2) "three"
3) "three"
4) "three"
5) "one"
redis>
```

Slika 19. Srandmember naredba

Naravno, postoje i naredbe koje vrše operacije koje se mogu samo vršiti nad skupovima, kao što su *sinter, sunion* i *sdiff*. Naredba *sinter* vrši operaciju presjeka nad skupovima, što znači da vraća sve članove koji su u jednom i drugom skupu. Da bi se naredba uspješno izvršila, potrebno je navesti dva skupa za parametre.

```
redis> SADD key1 "a"
(integer) 1
redis> SADD key1 "b"
(integer) 1
redis> SADD key1 "c"
(integer) 1
redis> SADD key2 "c"
(integer) 1
redis> SADD key2 "d"
(integer) 1
redis> SADD key2 "e"
(integer) 1
redis> SINTER key1 key2
1) "c"
redis>
```

Slika 20. Sinter naredba

Kao što vidimo na gornjoj slici, jedan skup ima članove “a”, “b” i “c”, dok drugi ima članove “c”, “d” i “e”. Kada izvršimo naredbu *sinter* nad tim skupovima, Redis vraća “c” jer je to jedini član koji je prisutan u oba skupa. Naredba *sunion* radi kao unija, što znači da vraća sve jedinstvene članove koji se nalaze u oba skupa. Opet, potrebno je navesti dva skupa kao parametre.

```
redis> SADD key1 "a"
(integer) 1
redis> SADD key1 "b"
(integer) 1
redis> SADD key1 "c"
(integer) 1
redis> SADD key2 "c"
(integer) 1
redis> SADD key2 "d"
(integer) 1
redis> SADD key2 "e"
(integer) 1
redis> SUNION key1 key2
1) "a"
2) "c"
3) "e"
4) "d"
5) "b"
redis>
```

Slika 21. Sunion naredba

Imamo jednake skupove kao i u prošlom primjeru. Kada se naredba uspješno izvrši, Redis vraća članove “a”, “b”, “c”, “d” i “e”. Pošto naredba vraća samo jedinstvene članove, “c” koji je bio u oba skupa je samo jednom prikazan. Naredba *sdiff* funkcionira kao razlika između skupova. Razlika vraća sve članove koji su u prvom skupu, a nisu u drugom.

```
redis> SADD key1 "a"
(integer) 1
redis> SADD key1 "b"
(integer) 1
redis> SADD key1 "c"
(integer) 1
redis> SADD key2 "c"
(integer) 1
redis> SADD key2 "d"
(integer) 1
redis> SADD key2 "e"
(integer) 1
redis> SDIFF key1 key2
1) "b"
2) "a"
redis>
```

Slika 22. Sdiff naredba

Opet imamo iste skupove kao i u prošlim primjerima. Nakon uspješnog izvršenja, Redis vraća članove “a” i “b”. Pošto je član “c” također i u drugom skupu, Redis ga ne vraća. Također, postoje inačice ovih naredbi koje vraćaju rezultate u novi skup. To su *sunionstore, sinterstore* i *sdiffstore* [17]. Svaka od ovih naredbi rade na istom princip, tako da ćemo prikazati samo jedan primjer. Ove naredbe imaju tri parametra, prvi je naziv novog skupa u koji stavljamo rezultate, a drugi i treći su opet skupovi nad kojima se operacija izvršava .

```
redis> SADD key1 "a"
(integer) 1
redis> SADD key1 "b"
(integer) 1
redis> SADD key1 "c"
(integer) 1
redis> SADD key2 "c"
(integer) 1
redis> SADD key2 "d"
(integer) 1
redis> SADD key2 "e"
(integer) 1
redis> SDIFFSTORE key key1 key2
(integer) 2
redis> SMEMBERS key
1) "b"
2) "a"
redis>
```

Slika 23. Sdiffstore naredba

Imamo situaciju sličnu kao i kod primjera sa *sdiff* naredbom. Nakon izvršenja naredbe *sdiffstore,* Redis vraća broj članova koji su ubačeni u novi skup. Opet su članovi “a” i “b” jedini

koji su u prvom skupu, a nisu u drugom. Sada možemo pomoću naredbe *smembers* potvrditi jesu li stvarno ti članovi prisutni u novom skupu. Redis vraća te članove, što znači da jesu.

3.1.5. Sortirani skupovi

Također, postoje i sortirani skupovi. Sortirani skupovi su vrlo slični običnim skupovima. Razlika je u tome što svaki član sortiranog skupa ima svoj broj bodova (engl. *score).*

```
> zadd hackers 1940 "Alan Kay"
(integer) 1
> zadd hackers 1957 "Sophie Wilson"
(integer) 1
> zadd hackers 1953 "Richard Stallman"
(integer) 1
> zadd hackers 1949 "Anita Borg"
(integer) 1
> zadd hackers 1965 "Yukihiro Matsumoto"
(integer) 1
> zadd hackers 1914 "Hedy Lamarr"
(integer) 1
> zadd hackers 1916 "Claude Shannon"
(integer) 1
> zadd hackers 1969 "Linus Torvalds"
(integer) 1
> zadd hackers 1912 "Alan Turing"
(integer) 1
```

Slika 24. Zadd naredba

Kao što vidimo na gornjoj slici, svaki član sortiranog skupa je asociran s brojem bodova. Da bi kreirali sortirani skup ili dodali članove u njega, koristimo naredbu *zadd* [17]. Naredbi dajemo dva parametra, broj bodova i ime člana. Da bi vratili određene članove sortiranog skupa, koristimo naredbu *zrange.* Prvi parametar naredbe je ime skupa, a drugi i treći određuju koji će se sve članovi skupa vratiti. Drugi i treći parametar mogu biti indeksi, brojevi bodova ili leksikografski poredak [17]. Ako su oni indeksi, možemo dati 0 i –1 da bi vratili sve članove skupa.

```
> zrange hackers 0 -1
1) "Alan Turing"
2) "Hedy Lamarr"
3) "Claude Shannon"
4) "Alan Kay"
5) "Anita Borg"
6) "Richard Stallman"
7) "Sophie Wilson"
8) "Yukihiro Matsumoto"
9) "Linus Torvalds"
```

Slika 25. Zrange naredba

Ako koristimo brojeve bodova, Redis će poredati sve članove koje imaju broj bodova između tih parametara. Ako koristimo leksikografski poredak, onda će se članovi poredati prema abecedi. Ako želimo poredati članove u obrnutom smjeru, koristimo *zrevrange* [17].

```
> zrevrange hackers 0 -1
1) "Linus Torvalds"
2) "Yukihiro Matsumoto"
3) "Sophie Wilson"
4) "Richard Stallman"
5) "Anita Borg"
6) "Alan Kay"
7) "Claude Shannon"
8) "Hedy Lamarr"
9) "Alan Turing"
```

Slika 26. Zrevrange naredba

Također, možemo dodati nastavak *withscore* da nam se pri vraćanju članova prikaže i njihov broj bodova. Moguće je i vraćanje ranga pojedinih članova skupa pomoću naredbe *zrank* [17].

```
> zrank hackers "Anita Borg"
(integer) 4
```

Slika 27. Zrank naredba

3.1.6. HyperLogLogs

Postoji još jedan tip podatka, temeljen na vjerojatnosti, a to je *HyperLogLogs*. On se koristi za brojanje jedinstvenih elemenata. Inače, za brojanje jedinstvenih elemenata je potrebna količina memorije jednaka broju tih elemenata jer memorija treba zapamtiti sve elemente koje je prošla. To nije slučaj s *HyperLogLogs*. Pri korištenju *HyperLogLogs,* memorija

koja se koristi pri brojanju jedinstvenih elemenata je konstantna, u najgorem slučaju oko 12.000 bajtova. Nedostatak ovog algoritma je to što vraća broj jedinstvenih elemenata sa standardnom greškom koja iznosi oko 1% [17]. Dodavanje elemenata se radi pomoću naredbe *pfadd.* Ona funkcionira jednako kao i kod dodavanja elemenata u listu.

```
redis> PFADD hll foo bar zap
(integer) 1
redis> PFADD hll zap zap zap
(integer) 0
redis> PFADD hll foo bar
(integer) 0
redis> PFCOUNT hll
(integer) 3
```

Slika 28. Pfadd i pfcount naredba

Broj jedinstvenih elemenata provjeravamo s naredbom *pfcount*. Na gornjoj slici vidimo da smo više puta dodali isti element s naredbom *pfadd* te Redis na kraju vraća broj 3, što i je točan broj jedinstvenih elemenata [17].

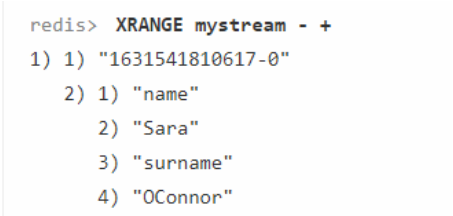
3.1.7. Tokovi

Naposljetku, imamo Redisove tokove (engl. *streams*). Koriste se za izgradnju posrednika za poruke, redova poruka, zapisnike i sustave za razgovor koji mogu čuvati povijest. U tokove je implementiran koncept grupe potrošača (engl. *consumer groups*). Cilj tih grupa je da omogući skupini klijenata suradnju pri konzumiranju različitih dijelova istih tokova. Potrošači u grupi mogu tražiti podatke prema njihovom ID-u, potvrditi obradu nekog podatka ili proglasiti vlasništvo nad porukom na čekanju [18]. Tokovi su slični listama na koje se može samo dodavati sadržaj. Svaki upis u tok ima svoj ID koji Redis automatski generira te svoju vrijednost, a dodaje se pomoću naredbe *xadd* [18].

```
redis> XADD mystream * name Sara surname OConnor
"1631541810617-0"
```

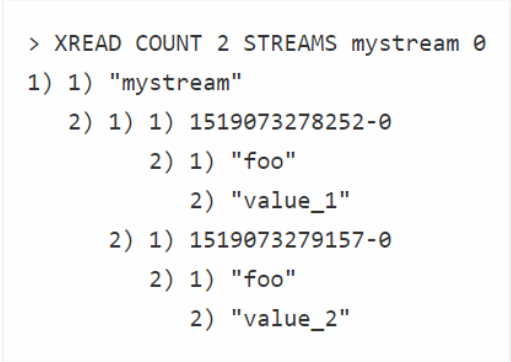
Slika 29. Xadd naredba

Prvi parametar naredbe je naziv toka kojem se upis dodaje, drugi je ID koji se može ručno specificirati ili automatski generirati ako je znak * stavljen na to mjesto te se dalje navode ključ - vrijednost parovi. Ako je korišten znak *, onda Redis vraća automatski generiran ID ako se naredba uspješno izvršila. Da bi dohvatili zapise tokova koriste se naredba *xrange* [18].



Slika 30. Xrange naredba

Kao i sa svim naredbama ovakvog tipa, potrebno je navesti ime toka kao prvi parametar te raspon ID-a kao drugi i treći parametar. No, za razliku od drugih sličnih naredbi, ako želimo dohvatiti sve zapise u nekom toku, koriste se - i + znakovi kao drugi i treći parametar. Naredba koja nudi mogućnost slušanja za nove poruke koje stižu u tok je *xread* [18].

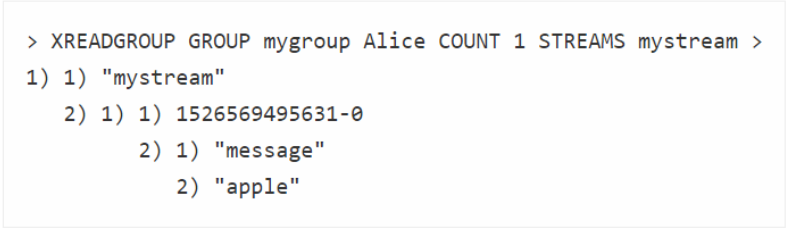


Slika 31. Xread s COUNT i STREAMS

Prvi parametar je COUNT koji ograničava broj zapisa koje se vraćaju naredbom i nije obavezan. Drugi parametar je STREAMS i on određuje koja je najmanja vrijednost ID-a zapisa kojeg želimo vratiti naredbom, a specificira se na kraju naredbe i obavezan je. Treći parametar je ime toka [18]. Naravno, potrebna je i naredba za kreiranje grupa, *xgroup create*.

Slika 32. XGROUP CREATE naredba

Potrebno je navesti naziv toka kojem pripada grupa kao prvi parametar i naziv grupe kao drugi parametar. Naposljetku, imamo naredbu *xreadgroup* koja služi za čitanje kao grupa potrošača [18].



Slika 33. XREADGROUP naredba

Prvi parametar GROUP specificira ime grupe, drugi parametar je ime osobe koja izvršava naredbu, treći parametar COUNT ograničava broj vraćenih zapisa i zadnji je ime toka. Naravno, ovdje su opisane samo osnove tokova. Redisovi tokovi su vrlo složeni tipovi podataka i za njihov potpun opis bilo bi potrebno puno više teksta.

3.2. Slučajevi korištenja

Kao što smo već rekli, nerelacijske baze podataka imaju mnogo specifičnih slučajeva korištenja, pa tako i Redis. U ovom dijelu proći ćemo kroz nekoliko različitih slučajeva korištenja i objasniti zašto se baš Redis koristi u tim slučajevima.

3.2.1. Predmemoriranje

Predmemoriranje (engl. *caching)* je proces spremanja podataka u radnu memoriju kako bi se ti podaci kasnije mogli brže dohvatiti. Predmemoriranje se može iskoristiti na mnogo korisnih načina, ali se kod Redisa najčešće koriste za aplikacijsko predmemoriranje pri:

- Pohrani DBMS podataka - Većina tradicionalnih baza podataka su danas dizajnirane za pružanje funkcionalnosti, dok je Redis dizajniran za pružanje brzine te ovdje predmemoriranje igra glavnu ulogu. Najčešći podaci baze koji se spremaju u predmemoriju su kopije tablice za pretraživanje i odgovori na velike upite iz DBMS-a. Ovo je korisno za poboljšavanje performansi aplikacije i smanjenje opterećenja na izvore podataka [22].
- Pohrani sesije od korisnika - Predmemoriranje podataka o sesiji korisnika je bitno za izgradnju dobrog korisničkog iskustva te responzivne aplikacije. Budući da gotovo svaka interakcija korisnika s aplikacijom zahtjeva pristup podacima u sesiji, spremanje tih podataka u predmemoriju drastično ubrzava responzivnost aplikacije. Također, predmemoriranje podataka iz sesije omogućava bilo kojem poslužitelju da obrađuje zahtjeve korisnika bez gubitka njegovog stanja [22].
- Pristupanju API odgovorima - Suvremene aplikacije su izgrađene na komponentama koje međusobno komuniciraju pomoću API-ja. Te komponente koriste API-je da bi napravile zahtjev prema drugim komponentama. Spremanje odgovora tih API-ja poboljšava performanse tih aplikacija, čak i kada se oni spremaju na kratko [22].

Predmemoriranje je također jedno od razloga zašto Redis ima vrlo impresivne performanse te zašto ga programeri koriste općenito. Predmemoriranje se može, na primjer, koristiti pri

spremanju korisničke košarice u *online* trgovinama te spremanju korisničkih postavki na stranici. To ćemo i pokazati kroz praktični primjer.

3.2.2. Natjecateljske ljestvice

U današnje vrijeme postoje vrlo popularne igre koje imaju natjecateljske ljestvice koje sadrže do čak više milijuna igrača. Prema Redisu [21], natjecateljske ljestvice ovakvih razmjera predstavljaju ogromne poteškoće iz različitih razloga:

- Ljestvica se ažurira kada dođe do bilo kakve promjene nad njom
- Korisnici mogu pregledavati ljestvice na više različitih načina i filtrirati podatke
- Korisnici mogu podijeliti svoje rezultate na socijalnim medijima
- Pružanje pristupa natjecateljskoj ljestvici u stvarnom vremenu te s visokom dostupnošću

Postoji više razloga nego što je ovdje navedeno, ali ovo su najznačajniji. Redisovo rješenje je upotreba sortiranih skupova pomoću kojih su natjecateljske ljestvice jednostavne za kreirat i upravljat. Sortirani skupovi su odlični za natjecateljske ljestvice jer:

- Sadrže jedinstvene članove
- Po prirodi su sortirani, samo je potrebno definirati broj bodova korisnika
- Bodovi se lako ažuriraju
- Ne zauzimaju puno memorije, tako da mogu sadržavati puno zapisa
- Zbog Redisovih performansi dohvaćanje zapisa nije toliki problem čak i kod velikog broja zapisa

Zbog ovih svih prednosti Redis je odličan izbor za aplikacije koje u sebi žele imati implementiranu natjecateljsku ljestvicu s velikim brojem korisnika.

3.2.3. Slanje poruka

Kao što smo već spomenuli, Redis se također koristi kao posrednik za poruke (engl. *Message broker*). Posrednik za poruke omogućuje aplikacijama, uslugama i sustavima međusobnu komunikaciju i razmjenu informacija, a to čini tako da prevodi poruke koje su poslane između protokola za razmjenu poruka [25]. No, Redis se može koristiti i za više od ovoga. Redis se može koristiti za sobe za dopisivanje visokih performansi, tokove (engl. *stream*) komentara u stvarnom vremenu te za sažetke društvenih medija (engl. *social media feed)* (Engel, 2017.). Prema Redisu [20], postoje tri razloga zašto je dobar izbor za slanje poruka:

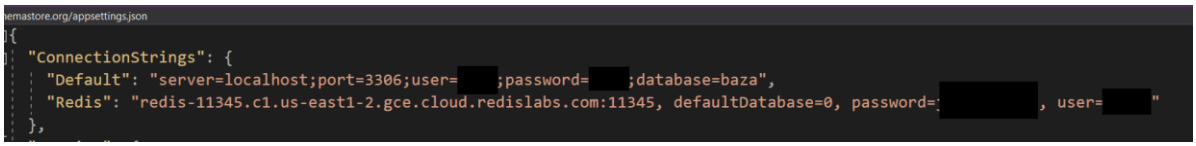
- Tokovi (engl. *streams)* - Idealni su za izgradnju posrednika za poruke, redova poruka, zapisnike i sustave za razgovor koji čuvaju povijest, za razliku od Pub/Sub poruka.

- Publish/Subscribe (hrv. *objavi/pretplati se*) - Protokol za razmjenu poruka koji omogućava da se klijent pretplati (engl. *subscribe)* na neki kanal. Kada taj kanal nešto objavi (engl. *publish*), tada svi klijenti koji su pretplaćeni na taj kanal dobivaju tu poruku, što je dobra potpora za društvene medije.
- Liste i sortirane liste – Mogu se koristiti za redove čekanja, a u ovom kontekstu se koriste za redove poruka.

Naravno, ovo nisu svi slučajevi korištenja za koje se Redis može iskoristiti, ali su među najpopularnijima. Kao što smo već rekli, slučajevi korištenja nerelacijskih baza podataka su prilično specifični, ali su sustavi tih baza podataka odličan izbor za takve slučajeve, kao što se može zaključiti iz ove cjeline.

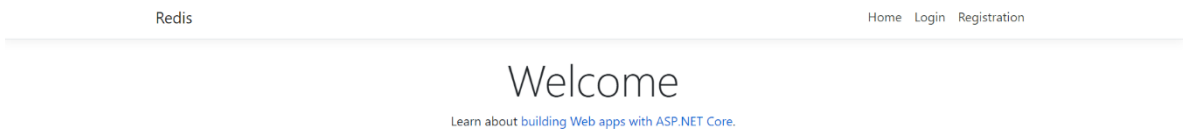
4. Praktični primjer

Za praktični primjer odlučeno je napraviti *online* trgovinu video igara. Aplikacija je napisana u .NET Core-u i koristi Redisov StackExchange *driver* kako bi aplikacija mogla komunicirati s Redisovim poslužiteljem. Aplikacija koristi Redis kao sekundarnu bazu podataka za realizaciju košarice pri kupnji video igara, pamćenju neuspjelih prijava korisnika te pamćenju korisničkih postavki na stranici. Za primarnu bazu podataka koristi se MySQL, u kojoj su spremljeni podaci o korisnicima, ulogama i igrama. Kako bi spajanje na baze podataka bilo moguće, potrebno je u datoteci "appsettings.json" dodati "ConnectionStrings" koji specificiraju sve što je potrebno da bi aplikacija imala pristup bazama.



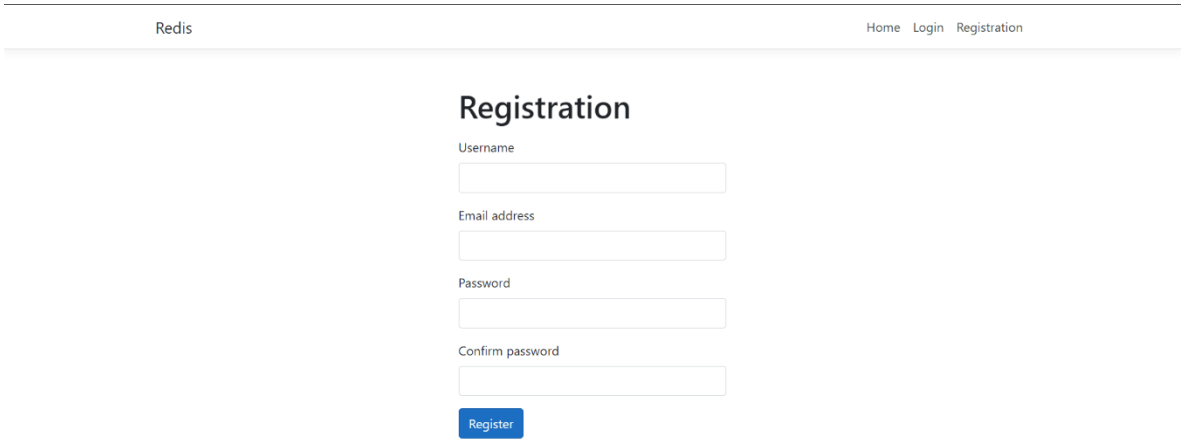
Slika 34. Appsettings.json postavke

Prvi znakovni niz specificira zahtjeve potrebne za spajanje na MySQL bazu podataka, a drugi za spajanje na Redis bazu podataka. Pri pokretanju aplikacije dolazimo na početnu stranicu koja samo služi za navigiranje korisnika.



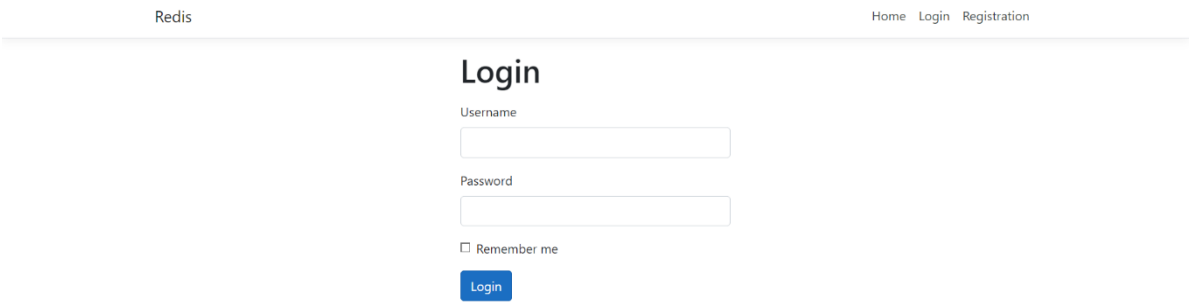
Slika 35. Početna stranica

Na početnoj stranici imamo samo izbor prijaviti se ili registrirati se. Idemo se prvo registrirati. Na stranici registracije moramo popuniti četiri polja: korisničko ime (engl. *username*), email adresa, lozinka i polje za potvrđivanje lozinke.



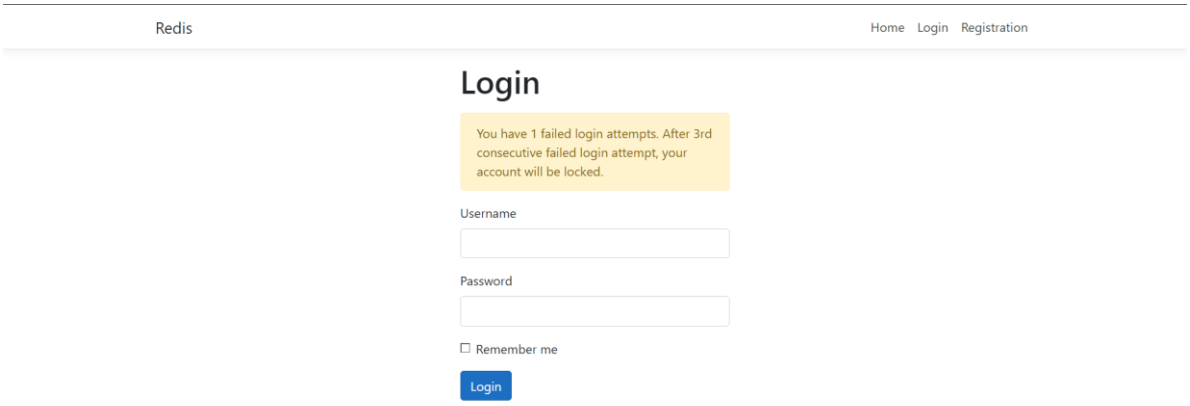
Slika 36. Stranica za registraciju

Kada se uspješno registriramo, možemo ići na stranicu za prijavu. Na stranici za prijavu trebamo samo upisati korisničko ime i lozinku kako bi uspješno prijavili.



Slika 37. Stranica za prijavu

No, u slučaju da se pokušamo prijaviti s krivim podacima, aplikacija će izbaciti sljedeću poruku: “Imate jedan neuspjeli pokušaj prijave. Nakon trećeg uzastopnog neuspjelog pokušaja, vaš će račun biti zaključan”.



Slika 38. Neuspjeli pokušaj prijave

Ovdje dolazimo do prve primjene Redisa u aplikaciji. Kada se korisnik neuspješno prijavi, pokreće se funkcija IncrementFailedAttempts koja prima za parametar korisničko ime tog korisnika.

```
1 reference
public async Task<int> IncrementFailedAttempts(string username)
{
    string key = username + "fa";

    return (int) await _redisDb.StringIncrementAsync(key);
}
```

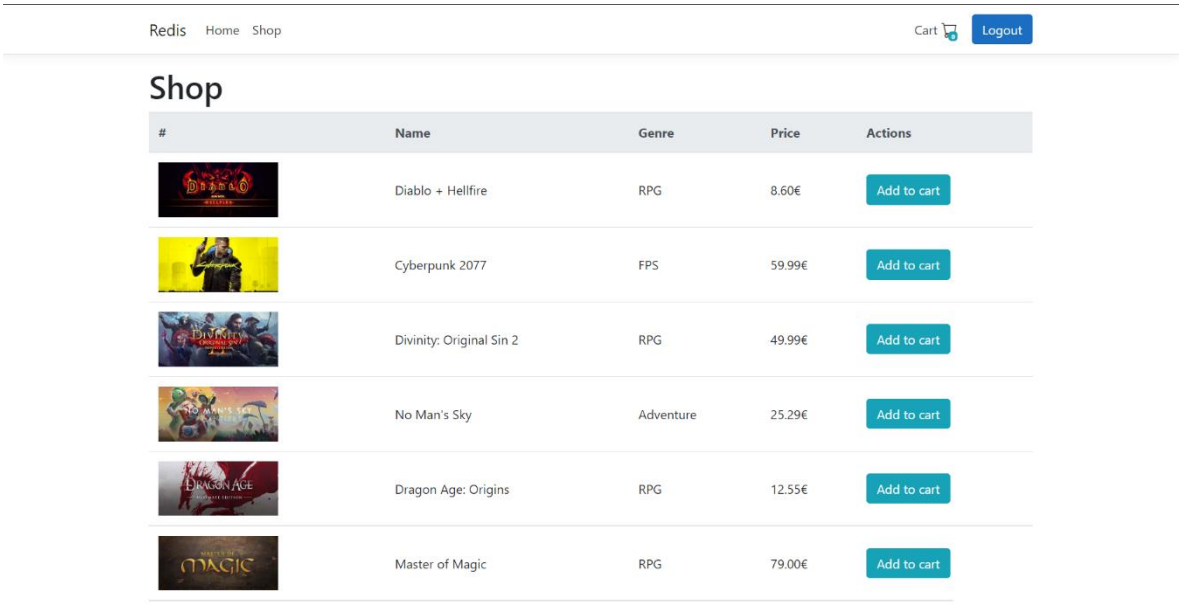
Slika 39. IncrementFailedAttempts funkcija

Funkcija uzima ime tog korisnika, sprema ga u znakovni niz *"key"* te još nadodaje znakovni niz "fa" (engl. *Failed attempt*) na njega. Taj znakovni niz šalje se Redisu sa naredbom *incr*. Ako taj ključ već postoji, onda će se njegova vrijednost za jedan za svaki neuspjeli pokušaj. Ako ključ ne postoji, onda će ga Redis kreirati i staviti njegovu vrijednost na 0 prije nego što ju poveća. Kada se korisnik uspije prijaviti dolazimo do funkcije ResetFailedAttempts.

```
1 reference
public async void ResetFailedAttempts(string username)
{
    string key = username + "fa";

    await _redisDb.KeyDeleteAsync(key);
}
```

Slika 40. ResetFailedAttempts funkcija

Funkcija radi na isti način kao i prethodna funkcija, samo što koristi Redisovu naredbu *getdel* kako bi obrisala broj neuspjelih prijava. Kada se korisnik uspješno prijavi, ima pristup trgovini.



Slika 41. Stranica za trgovinu

Trgovina sadrži tablični prikaz igara koje su dostupne za kupiti. Sve igre sadrže svoju sliku, ime, žanr i cijenu. Također, igre se mogu dodati u košaricu, što je glavna funkcionalnost Redisa u aplikaciji. Kada pritisnemo gumb “Add to cart”, pokreće se funkcija AddToCart.

```
[HttpPost]
0 references
public async Task<IActionResult> AddToCart([FromBody] CreateUserGame userGame)
{
    string username = _userManager.GetUserName(User);

    var listLength = await _redisDb.ListRightPushAsync(username, userGame.id);

    return Ok(listLength);
}
```

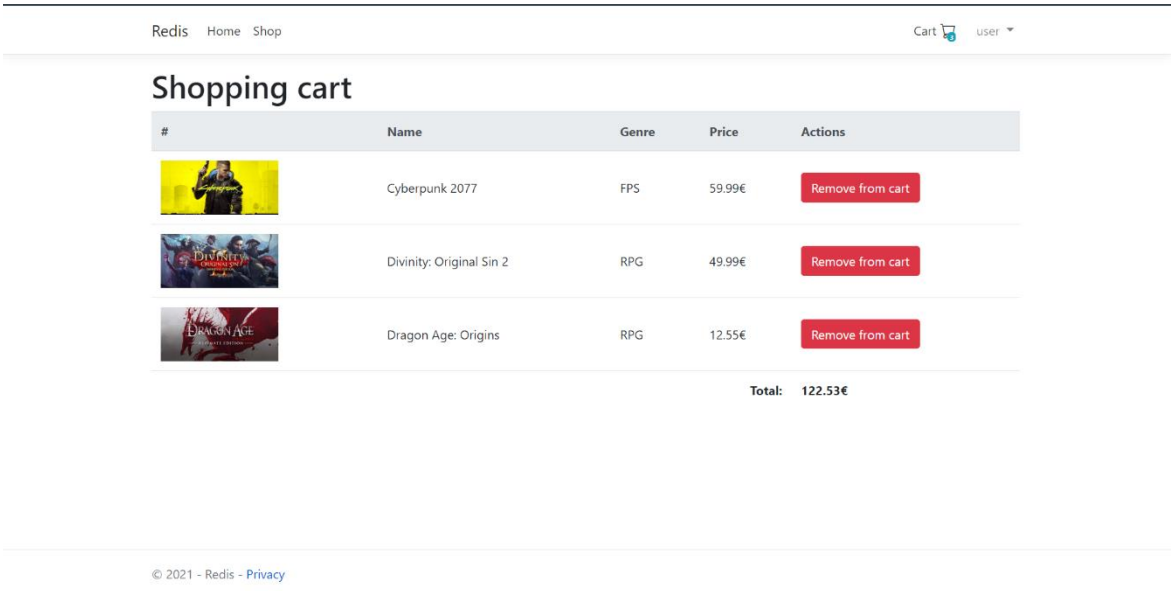
Slika 42. AddToCart funkcija

Funkcija AddToCart uzima korisničko ime i ID igre koja se želi dodati u košaricu i pomoću Redis naredbe *rpush* ju dodaje u listu. Lista će poprimiti naziv imena korisnika, tako da svaki korisnik ima svoju zasebnu košaricu jer je korisničko ime jedinstveno, a vrijednost koja se zapisuje u listu je ID igre. U gornjem desnom kutu stranice nalazi se ikonica s košaricom koja nam prikazuje koliko ona sadrži artikala. Za prikazivanje tog broja također se koristi Redis, pomoću funkcije Cart.

```
0 references
public async Task<IActionResult> Cart(ShopCartViewModel model)
{
    if (ModelState.IsValid)
    {
        List<Game> itemsInCart = new List<Game>();

        string username = _userManager.GetUserName(User);
        string key = username + "dm";

        int listLength = (int) await _redisDb.ListLengthAsync(username);
        int darkMode = (int)await _redisDb.StringGetAsync(key);

        for (int i = 0; i < listLength; i++)
        {
            int gameId = (int) await _redisDb.ListGetByIndexAsync(username, i);
            var game = _context.Games.FirstOrDefault(game => game.Id == gameId);

            if (game != null)
            {
                itemsInCart.Add(game);
            }
        }

        model.CartItems = itemsInCart;
        model.DarkTheme = (darkMode == 1) ? true : false;
        model.ItemsCounter = itemsInCart.Count;
    }

    return View(model);
}
```

Slika 43. Cart funkcija

Funkcija uzima korisničko ime da bi provjerila koliko ima elemenata u istoimenoj listi koja sadrži igre spremljene u košaricu. Funkcija uzima taj broj i sprema je u varijablu “listLength”.
Klikom na ikonicu dolazimo na stranicu košarice koja nam daje tablični prikaz svih igara kojih smo dodali u košaricu.



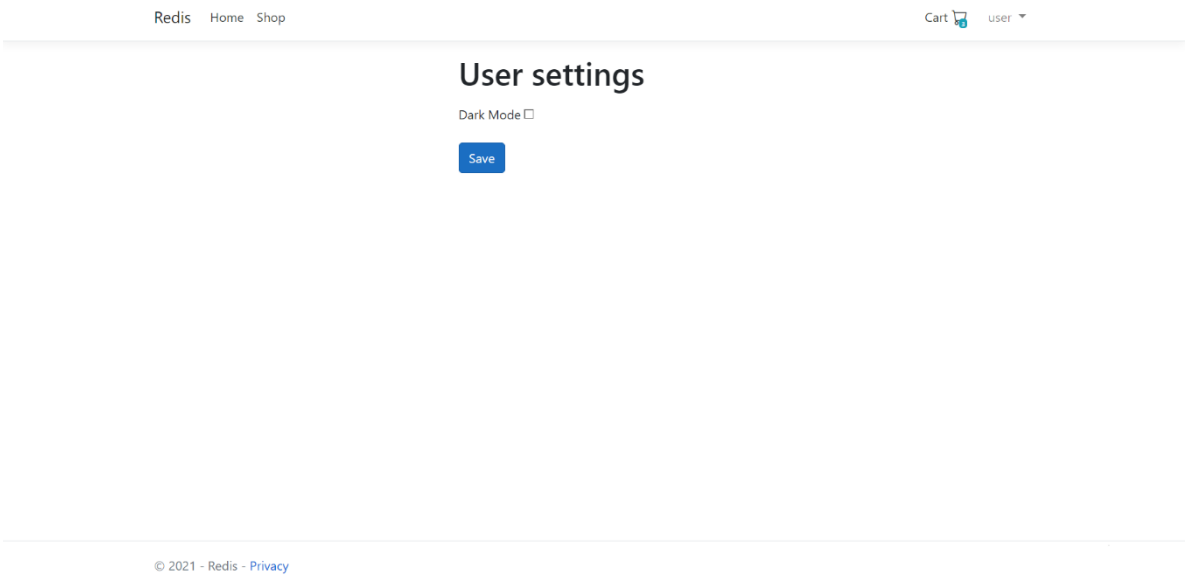
Slika 44. Stranica za košaricu

Ovdje imamo izbor brisanja igara iz košarice, što možemo napraviti pritiskom na gumb “Remove from cart”. Kada pritisnemo taj gumb, pokreće se funkcija RemoveFromCart.

```
[HttpPost]
0 references
public async Task<IActionResult> RemoveFromCart([FromBody] RemoveUserGame userGame)
{
    string username = _userManager.GetUserName(User);

    var listLength = await _redisDb.ListRemoveAsync(username, userGame.id, 1);

    return Ok(listLength);
}
```

Slika 45. RemoveFromCart funkcija

Funkcija RemoveFromCart vrlo je slična funkciji AddToCart, razlika je u tome što funkcija RemoveFromCart uzima korisničko ime i ID igre da bi izbrisao zapis te igre iz liste korisnika. Naposljetku, imamo postavke korisnika koje se nalaze u gornjem desnom kutu pored košarice.



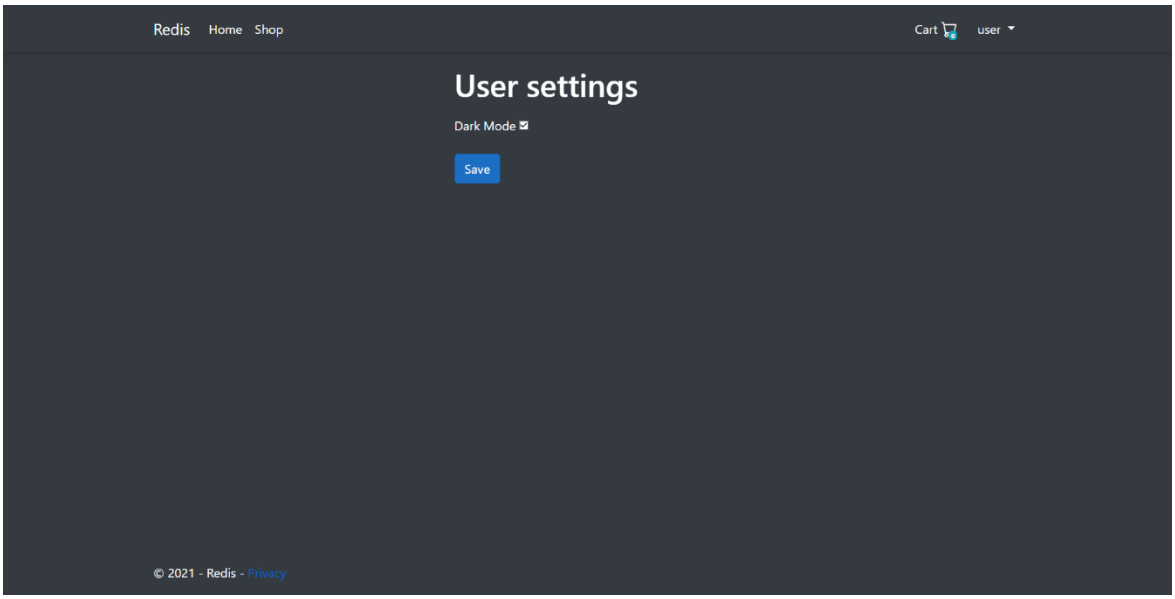
Slika 46. Stranica za korisničke postavke

Ovdje imamo samo jednu postavku, ali poanta je pokazati kako je s Redisom moguće spremiti postavke korisnika na stranici. Kada označimo postavku “Dark Mode” i kliknemo na gumb “Save”, pokreće se funkcija SaveSettings.

```
[HttpPost]
0 references
public async Task<IActionResult> SaveSettings(SettingsViewModel model)
{
    if (ModelState.IsValid)
    {
        string username = _userManager.GetUserName(User);
        string key = username + "dm";

        int val = model.DarkTheme ? 1 : 0;

        await _redisDb.StringSetAsync(key, val);
    }

    return View("Settings", model);
}
```

Slika 47. SaveSettings funkcija

Funkcija uzima korisničko ime i dodaje na njega znakovni niz “dm” te je sprema u varijablu “key”. Zatim uzima broj 1 ili 0 ovisno o tome jel označena postavka za “Dark Mode” te je sprema u varijablu “val”. Nakon toga Redisu šalje naredbu *set* za postavljanje znakovnog niza s ključem čiji je naziv jednak varijabli “key” te vrijednosti jednakoj varijabli “val”.



Slika 48. Mračna tema stranice

Kada spremimo postavke, ako smo označili “Dark Mode”, sve stranice će imati mračnu temu.

5. Zaključak

Glavni cilj ovog rada je bio dokazati da su nerelacijske baze podataka vrijedne za razmotriti i koristiti pri početku projekta te dokazati to kroz cjeline i praktični primjer. U prvoj cjelini prošli smo kroz povijest nerelacijskih baza podataka i njihove primjene te razvoj kroz vrijeme. Usporedili smo glavna obilježja relacijskih i nerelacijskih baza podataka, spomenuli u kojim slučajevima su ta obilježja pogodna, naveli prednosti i nedostatke kroz koje smo zaključili da prednosti nerelacijskih baza podataka nadmašuju njihove nedostatke.

U drugoj cjelini opisivali smo Redis, osnove korištenja njime, sintaksu i njegove tipove podataka. Kroz tipove podataka prikazali smo jednostavnost korištenja Redisa i koliko je intuitivan za naučiti. Također, pokazali smo korisnost tih tipova podataka i potkrijepili to kroz nekoliko najpopularnijih slučajeva korištenja Redisa. Kroz slučajeve korištenja smo također dokazali da nerelacijske baze podataka imaju specifičnu korist, ali i da su vrlo korisne u tim slučajevima.

U trećoj cjelini prošli smo kroz praktični primjer te objasnili kako radi aplikacija. Za izradu praktičnog primjera koristili smo Visual Studio i .NET CORE programski jezik, MySQL Workbench za primarnu bazu podataka i, naravno, Redis za sekundarnu bazu podataka. Kroz slike i objašnjenja koda smo demonstrirali koliko je jednostavno koristiti Redis u programskom kodu, gotovo sve implementacije Redisa napravljene su u jednoj liniji koda. Kada sve rezimiramo, mislim da smo uspjeli dokazati ono što htjeli u svim cjelinama te ustanovili da je Redis odličan suvremeni sustav za upravljanje baza podataka.

.

6. Popis literature

[1] K. D. Foote, “A Brief History of Non-Relational Databases”, 2018. [Na internetu]. Dostupno: https://www.dataversity.net/a-brief-history-of-non-relational-databases/# [pristupano 28.8.2021.].

[2] A. Pore, “How to Choose the Right NoSQL Database for Your Application?”, 2018. [Na internetu]. Dostupno: http://www.dataversity.net/choose-right-nosql-database-application/ [pristupano 28.8.2021.].

[3] “NoSQL” (bez dat.) u Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/NoSQL [pristupano 29.8.2021.].

[4] IBM Cloud Education, “CAP Theorem”, 2019. [Na internetu]. Dostupno: https://www.ibm.com/cloud/learn/cap-theorem [pristupano 29.8.2021.].

[5] A. Scholes, “DISAMBIGUATING ACID AND CAP”, 2015. [Na internetu]. Dostupno: https://www.voltdb.com/blog/2015/10/disambiguating-acid-cap/ [pristupano 29.8.2021.].

[6] “Key–value database” (bez dat.) u Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/Key–value_database [pristupano 29.8.2021.].

[7] “Document–oriented database” (bez dat.) u Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/Document-oriented_database [pristupano 30.8.2021.].

[8] “Wide–column store” (bez dat.) u Wikipedia, the Free Encyclopedia. Dostupno: https://en.wikipedia.org/wiki/Wide-column_store [pristupano 30.8.2021.].

[9] G2, “Best Graph Databases” (bez dat.). [Na internetu] Dostupno: https://www.g2.com/categories/graph-databases [pristupano 31.8.2021.].

[10] K. D. Foote, “A Review of Different Database Types: Relational versus Non-Relational", 2016. [Na internetu]. Dostupno: http://www.dataversity.net/review-pros-cons-different-databases-relational-versus-non-relational/ [pristupano 1.9.2021.].

[11] M. Berga, T. Franco, “SQL VS NOSQL: WHEN TO USE?”, 2021. [Na internetu]. Dostupno: https://www.imaginarycloud.com/blog/sql-vs-nosql/ [pristupano 2.9.2021.].

[12] I. P. Chinedu, “SQL vs. NoSQL: The Differences Explained”, 2021. [Na internetu]. Dostupno: https://blog.panoply.io/sql-or-nosql-that-is-the-question [pristupano 2.9.2021.].

[13] B. Miller, “7 Pros and Cons of NoSQL”, 2017. [Na internetu]. Dostupno: https://greengarageblog.org/7-pros-and-cons-of-nosql [pristupano 3.9.2021.].

[14] A. Williams, “NoSQL Beginner Guide: Pros, Cons, Types, and Philosophy”, 2021. [Na internetu]. Dostupno: https://www.altexsoft.com/blog/nosql-pros-cons/ [pristupano 3.9.2021.].

[15] Talha, “Relational Database Vs NoSQL: A Comprehensive Analysis”, 2021. [Na internetu]. Dostupno: https://hevodata.com/learn/relational-database-vs-nosql-a-deep-analysis/ [pristupano 3.9.2021.]

[16] B. Anderson, B. Nicholson, “SQL vs. NoSQL Databases: What's the Difference?”, 2021. [Na internetu]. Dostupno: https://www.ibm.com/cloud/blog/sql-vs-nosql [pristupano 3.9.2021.].

[17] Redis (bez dat.) “An introduction to Redis data types and abstractions” [Na internetu]. Dostupno: https://redis.io/topics/data-types-intro [pristupano 4.9.2021.].

[18] Redis (bez dat.) “Introduction to Redis Streams” [Na internetu]. Dostupno: https://redis.io/topics/streams-intro [pristupano 9.9.2021.].

[19] J. Engel, “TOP 5 REDIS USE CASES”, 2017. [Na internetu]. Dostupno: https://www.objectrocket.com/blog/how-to/top-5-redis-use-cases/ [pristupano 10.9.2021.].

[20] Redis (bez dat.) “Messaging” [Na internetu]. Dostupno: https://redis.com/solutions/use-cases/messaging/ [pristupano 10.9.2021.].

[21] Redis (bez dat.) “Leaderboards” [Na internetu]. Dostupno: https://redis.com/solutions/use-cases/leaderboards/ [pristupano 10.9.2021.].

[22] Redis (bez dat.) “Caching” [Na internetu]. Dostupno: https://redis.com/solutions/use-cases/caching/ [pristupano 10.9.2021.].

[23] G. Harrison, “Eventual Consistency”, 2011. [Na internetu]. Dostupno: https://www.dbta.com/Columns/Applications-Insight/Eventual-Consistency-73004.aspx [pristupano 3.9.2021.]

[24] Redis (bez dat.) “Introduction to Redis” [Na internetu]. Dostupno: https://redis.io/topics/introduction [pristupano 4.9.2021.].

[25] IBM Cloud Education, “Message brokers”, 2019. [Na internetu]. Dostupno: https://www.ibm.com/cloud/learn/message-brokers [pristupano 10.9.2021.].

7. Popis slika

8. Popis tablica