OSIGURAVANJE VISOKE DOSTUPNOSTI BAZA PODATAKA KORIŠTENJEM MICROSOFT TEHNOLOGIJA

Gladoić, Vladimir

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

OSIGURAVANJE VISOKE DOSTUPNOSTI BAZA PODATAKA KORIŠTENJEM MICROSOFT TEHNOLOGIJA

Vladimir Gladoić

Zagreb, veljača 2019.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.

U Zagrebu, 12.02.2019.

Vladimir Gladović

Predgovor

Ovim putem zahvaljujem se svojoj obitelj, kolegama s posla i prijateljima na razumijevanju i potpori za vrijeme studiranja. Zahvaljujem se i svom mentoru mr. sc. Mariju Fabijaniću na ukazanom povjerenju te pomoći i savjetima tijekom izrade završnog rada.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

U ovom završnom radu će se nastojati prikazati kako osigurati visoku dostupnost baza podatka u kritičnim IT sustavima. Jedna od glavnih zadaća sustava za visoku dostupnost je minimizirati vrijeme zastoja (engl. *downtime*) baza podataka, a samim time i maksimizirati dostupnost. Stoga su u radu prikazana neka od rješenja za visoku dostupnost te je posebna pažnja posvećena *AlwaysOn* tehnologiji, koja je preporučeno i aktualno Microsoftovo rješenje, a osigurava nas u potpunosti od gubitka podataka te smanjuje vrijeme zastoja na minimum. U radu će se prikazati i analiza utjecaja dodavanja sekundarnih replika na performanse sustava te će biti proučen utjecaj dodavanja replike za čitanje na rasterećenje primarne replike na kojoj se izvršavaju ostale *CRUD* operacije. Kako bi se predočilo kako to u stvarnosti izgleda kada poslovna aplikacija koristi *SQL* bazu podatka, kojoj je osigurana visoka dostupnost pomoću *AlwaysOn* tehnologije, kreirana je *web* aplikacija koristeći *ASP.NET MVC* tehnologiju. Prikazat će se kako se prebacivanje baza (engl. *failover*) i povratno prebacivanje (engl. *failback*) odražava na rad aplikacije te kako se dijelovi aplikacije, koji samo čitaju podatke, mogu uz minimalne izmjene u kodu prebaciti na repliku za čitanje (engl. *read-only replica*). Također, bit će prikazano i kako se može uspostaviti distribucija opterećenja (engl. *load-balancing*) između više replika za čitanje. Cilj ovog završnog rada je opisati kako uspostaviti visoku dostupnost baza podataka i prikazati kako će to utjecati na performanse te kako se na to može utjecati dodavanjem sekundarnih replika, replika za čitanje te uspostavom distribucije opterećenja.

Ključne riječi: AlwaysOn, high availability, ASP.NET MVC, SQL, failover, failback, read-only replica, load-balancing

Summary

This final paper will attempt to demonstrate how to implement high-availability database in mission-critical environment. One of the main duties of high-availability systems is to minimize downtime of the database and maximize database availability. Some high-availability solutions are presented in the paper, but the main focus will be in *AlwaysOn*

technology which is recommended by Microsoft and prevents data loss and minimizes downtime. Paper also analyze the performance improvement gained by adding a read-only secondary replica which offloads *CRUD* operations from the primary server. For all tests and demonstrations database high availability shown in this paper, a web application was created using *ASP.NET MVC* technology. The demonstration will include an analysis how failover and failback impact the application, and how with minimal changes to application code we can transfer all read operations to the secondary replica. *Load-balancing* between replicas will also be shown.

The goal of this paper is to describe how to implement high-availability databases, show the performance improvement, adding a secondary replica, adding read-only replicas and using load-balancing in our environment.

Keywords: AlwaysOn, high availability, ASP.NET MVC, SQL, failover, failback, read-only replica, load-balancing

Sadržaj

1. Uvod

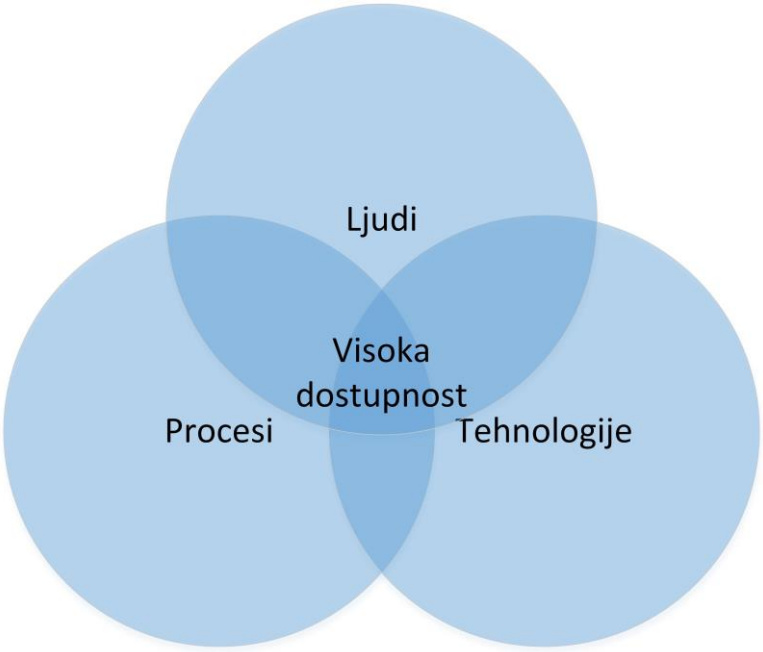
Ideja za pisanje ovog rada razvila se zbog najavljenih ukidanja zastarjelih tehnologija (*database mirroring*) iz *Microsoft SQL Servera* pa su trebale biti istražene najpouzdanije dostupne tehnologije visoke dostupnosti koje se nude na tržištu. Nakon usporedbe dostupnih tehnologija odluka je pala za *AlwaysOn* tehnologiju koja, ako se pravilno konfigurira, jamči da neće doći do gubitka podataka. Kako bi se objasnio princip rada tehnologija visoke dostupnosti te sustava oporavka od katastrofe, u drugom poglavlju će biti definirani osnovni pojmovi potrebni za razumijevanje tih tehnologija. Cilj ovog završnog rada jest prikazati koje su tehnologije visoke dostupnosti dostupne te napraviti usporedni prikaz prednosti i nedostataka svake od njih, što će detaljno biti opisano u trećem poglavlju. U četvrtom poglavlju će biti opisano kako implementirati *AlwaysOn* rješenje koje se smatra jednim od pouzdanijih i koje predstavlja trenutno aktualnu tehnologiju preporučenu od Microsofta. U petom poglavlju opisat će se kako konfigurirati sekundarne replike za čitanje. Za potrebe demonstracije rada s *AlwaysOn* grupama kreirana je i aplikacija koja predstavlja primjer poslovne aplikacije koja koristi *SQL Server* bazu podatka uz osiguranu visoku dostupnost te oporavak od katastrofe, što će biti opisano u šestom poglavlju. Kako bi se prikazalo kakav učinak ima osiguravanje visoke dostupnosti baza podataka te dodavanje sekundarnih replika na performanse sustava, u zadnjem poglavlju će biti opisana mjerenja provedena u tu svrhu te će biti prikazani rezultati mjerenja, uključujući i usporedni prikaz performansi prije i poslije dodavanja replika.

2. Koncepti tehnologije visoke dostupnosti i oporavka sustava od katastofe

2.1. Osnovni pojmovi

Glavni cilj visoke dostupnosti (engl. *high availability*, skraćeno HA) je minimizirati i ublažiti učinak nedostupnosti (engl. *downtime*) baza podataka za krajnjeg korisnika te maksimizirati dostupnost.[1]

Kako bi se mogao razumjeti pojam visoke dostupnosti, najprije će se definirati što to podrazumijeva visoka dostupnost, odnosno koje su sve komponente potrebne da bi se stvorilo visoko dostupan sustav. To su: ljudstvo, procesi i tehnologija. Ljudi moraju imati odgovarajuće vještine, znanja i odgovornost, procesi moraju biti dokumentirani te ih se mora slijediti i provoditi, a tehnologija treba ispunjavati poslovne zahtjeve za postizanje visoke dostupnosti (Slika 2.1).



Slika 2.1 Tri stupa visoke dostupnosti[2]

[1] Pro SQL Server Always On Availability Groups; Uttam Parui, Vivek Sanil; str. 3.
[2] Ibidem; str. 4.

Oporavljanje od katastrofe (engl. *Disaster Recovery*, skraćeno DR) čine procesi, politike i procedure koje pripremaju vitalnu informatičku infrastrukturu organizacije za učinkovit oporavak nakon što se desi prirodna katastrofa ili katastrofa koju je izazvao čovjek.[3]

Sustav može biti nedostupan iz više razloga; uzrok može biti mreža, aplikacija, hardver ili sama baza podataka, no ovaj rad će se ograničiti samo na dostupnost baza podataka te na mjerenje razine dostupnosti *SQL Servera.* Razina dostupnosti nekog sustava može se definirati kao količina vremena u kojemu je sustav dostupan krajnjem korisniku (engl. *uptime*)[4], koja se izražava u postotcima, a izračunava se formulom kao što je prikazano na slici (

Slika 2.2).

$$Dostupnost = \frac{Stvarno\ vrijeme\ neprekidnog\ rada}{Očekivano\ vrijeme\ neprekidnog\ rada}\ x\ 100\ \%$$

Slika 2.2 Formula za izračun dostupnosti[5]

Prema navedenoj formuli lako dolazimo do postotnog iznosa razine dostupnosti nekog sustava, a on se uobičajeno izražava u broju devetki, kako je prikazano na slici (Slika 2.3). Tako se uobičajeno u ugovorima o razini dostupnosti (engl. *service-level agreements*, skraćeno SLA) propisuje koliko neki servis može biti nedostupan u slučaju kvara.

Level of Availability	Downtime per Week	Downtime per Month	Downtime per Year
99%	1 hour, 40 minutes, 48 seconds	7 hours, 18 minutes, 17 seconds	3 days, 15 hours, 39 minutes, 28 seconds
99.9%	10 minutes, 4 seconds	43 minutes, 49 seconds	8 hours, 45 minutes, 56 seconds
99.99%	1 minute	4 minutes, 23 seconds	52 minutes, 35 seconds
99.999%	6 seconds	26 seconds	5 minutes, 15 seconds

Slika 2.3 Tablica razine dostupnosti[6]

[3] https://www.imperva.com/learn/availability/disaster-recovery/; 6.10.2019.
[4] SQL Server AlwaysOn Revealed, 2nd Edition; P. A. Carter; str. 1.
[5] Administering a SQL Database Infrastructure; Victor Isakov; str. 266.
[6] SQL Server AlwaysOn Revealed, 2nd Edition; P. A. Carter; str. 2.

Što su zahtjevi iz ugovora za brojem devetki u razini dostupnosti veći, to se u pravilu i troškovi eksponencijalno povećavaju, jer sve veća razina dostupnosti zahtijeva veću redundanciju na svim razinama infrastrukture (mreža, uređaji za pohranu podatka, poslužitelji u HA načinu rada).

2.2. Tehnologija visoke dostupnosti

Glavni zadatak tehnologija visoke dostupnosti je umanjiti rizike koje sa sobom donose greške u programu te potencijalni kvarovi hardvera. Kako bi se smanjili ti rizici, poslovno kritične aplikacije postavljaju se na hardver koji ima sve komponente redundantne kako bi osigurao otpornost na kvarove[7]. Tehnologije visoke dostupnosti temelje se na principu automatskog prebacivanja na redundantni sustav ako dođe do kvara ili problema na primarnom sustavu. S obzirom na to da je predmet ovog završnog rada upravo osiguranje visoke dostupnosti, u nastavku rada biti će detaljno opisane glavne tehnologije koje u tu svrhu pruža Microsoft.

2.3. Oporavak IT sustava od katastrofe

Iako je uspostavljen sustav visoke dostupnosti unutar jednog podatkovnog centra (engl. *data centar*), to nije garancija da poslovno kritične aplikacije (engl. *mission-critical applications*) neće postati nedostupne jer se može dogoditi da uslijed katastrofe većih razmjera (npr. poplava, potres, teroristički napad...) bude uništen cijeli podatkovni centar unutar kojeg je osigurana visoka dostupnost. Od takvih situacija nas spašavaju sekundarne replike podataka ili sigurnosne kopije na udaljenoj lokaciji. U tom slučaju se govori o oporavku od katastrofe (engl. *disaster recovery*, skraćeno DR).

Za svaki ozbiljniji IT sustav s poslovno kritičnim aplikacijama, potrebno je definirati dozvoljeni gubitak podataka (engl. *recovery point objective*, skraćeno RPO) i dozvoljenu nedostupnost sustava (engl. *recovery time objective*, skraćeno RTO). Oni moraju biti definirani u ugovoru o razini usluge (engl. *service-level agreements*, skraćeno SLA) te s promjenom okruženja moraju biti revidirani.

[7] Ibidem; str. 1.

RPO predstavlja maksimalnu prihvatljivu razinu izgubljenih podatka nakon nezgode. Predstavlja točku u vremenu u koju izgubljeni podaci moraju biti vraćeni.[8]

RTO predstavlja najveće dopušteno vrijeme nedostupnosti kada se dogodi kvar. To je maksimalna prihvatljiva količina vremena potrebna za ponovno pokretanje određenog poslovnog procesa, odnosno da se poslovanje vrati u normalu.[9]

Iako se rješenja za visoku dostupnost i za oporavak od katastrofe razlikuju, međusobno su i isprepletena. Rješenja za visoku dostupnost osiguravaju dostupnost podataka, dok rješenja za oporavak od katastrofe služe za oporavak podataka i nastavljanje visoke dostupnosti. Tako npr. *AlwaysOn* dostupne grupe mogu poslužiti kao rješenje za visoku dostupnost, ukoliko su replike smještene unutar istog podatkovnog centra te kao rješenje za oporavak od katastrofe, ukoliko se sekundarna replika nalazi u udaljenom podatkovnom centru.[10]

[8] Pro SQL Server Always On Availability Groups; Uttam Parui, Vivek Sanil; str.. 7.
[9] Ibidem; str. 7.
[10]https://docs.microsoft.com/en-us/sql/database-engine/sql-server-business-continuity-dr?view=sql-server-2017; 6.10.2019.

3. Postojeća rješenja visoke dostupnosti baza podataka

SQL Server nudi nekoliko rješenja za visoku dostupnosti i oporavak od katastrofe. Neka od njih su na razini instance, a neka na razini baze. Zajedničko im je to da u svim slučajevima postoji redundantna infrastruktura koja omogućava održavanje dostupnost aplikacija i servisa. Prema stanju pripravnosti (engl. *standby*), sva ta rješenja mogu se podijeliti u 3 kategorije: *Hot*, *Warm* i *Cold* .

Prema toj klasifikaciji *hot standby* poslužitelj je onaj koji je sinkroniziran s primarnim poslužiteljem te je konfiguriran tako da omogućuje automatsko prebacivanje baza podatka na sekundarni poslužitelj, i pogodan je kao rješenje visoke dostupnosti.

Warm standby poslužitelj je također sinkroniziran s primarnim poslužiteljem, ali nije konfiguriran za automatsko prebacivanje baza te je stoga pogodan kao rješenje za oporavak od katastrofe.

Cold standby poslužitelj nije sinkroniziran s primarnim poslužiteljem i stoga nije pogodan za automatsko prebacivanje baza.

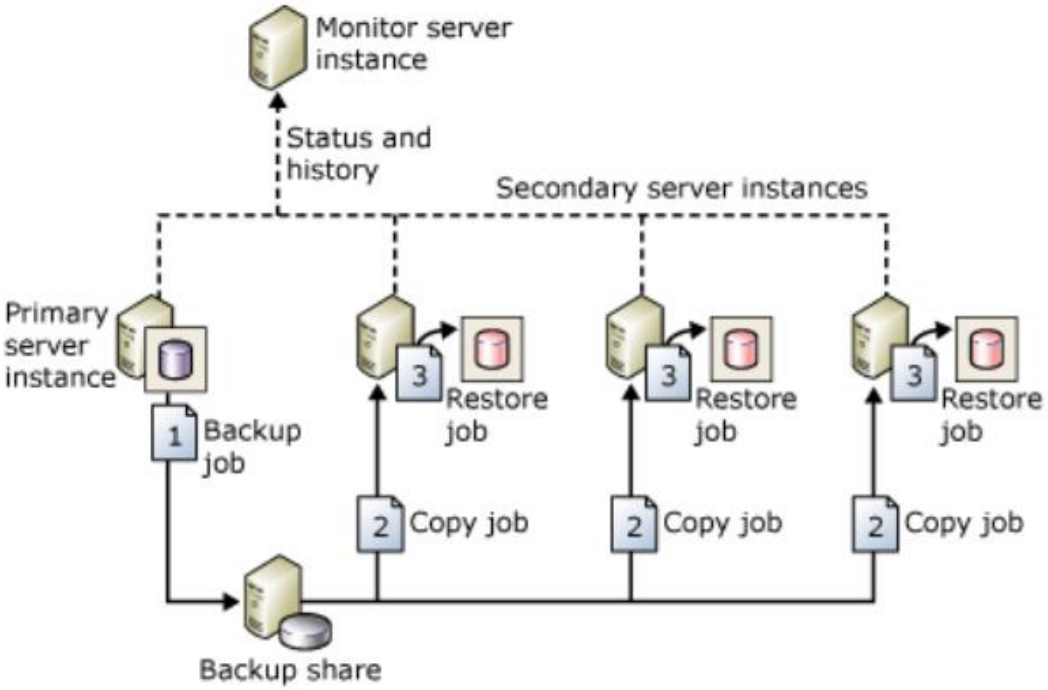
U ovom radu prikazat će se najzastupljenija rješenja visoke dostupnosti te prednosti i nedostaci svakog od njih.

3.1. Log shipping tehnologija

Log shipping je tehnologija koja se koristi za implementaciju oporavka od katastrofe, a djeluje na razini baze. Radi na principu izrade sigurnosne kopije (engl. *backup*) transakcijskih zapisnika (engl. *transaction logs*) na primarnom poslužitelju, njihovom kopiranju na sekundarni poslužitelj te ponovnom uspostavljanja (engl. *restore*) na sekundarnom poslužitelju ili poslužiteljima.

Jedan primarni poslužitelj (instanca) može imati više sekundarnih poslužitelja te jednu instancu za *monitoring,* kako je i prikazano na slici (

Slika 3.1).



Slika 3.1 Primjer *log shipping* konfiguracije[11]

Na gornjem primjeru *log shipping* konfiguracije postoji jedna instanca primarnog poslužitelja, tri instance sekundarnih poslužitelja te jedna monitor instanca. Za izvršenje koraka *log shippinga* zadužen je *SQL Agent* koji mora biti pokrenut na primarnoj i na sekundarnim instancama. Na primarnoj instanci pokreću se zadaci (engl. *jobs*) za izradu sigurnosne kopije transakcijskih zapisnika primarne baze na dijeljeni disk. Na sekundarnim poslužiteljima nalaze se zadaci za kopiranje sigurnosnih kopija transakcijskih zapisnika te zadaci za primjenu kopiranih transakcijskih zapisnika na sekundarnu bazu.

Prema klasifikaciji po stanju pripravnosti, *log shipping* spada u kategoriju *warm* rješenja. Često se koristi u slučajevima gdje postoji potreba za vremenskom odgodom primjene transakcijskih zapisnika na sekundarnoj lokaciji jer to nije moguće kod *AlwaysOn* tehnologije i visoko dostupnih grupa. To je čest slučaj kod implementacije oporavka od katastrofe, gdje je cilj zaštititi se od slučajnog brisanja podatka iz baze ili pojedine tablice. U tom slučaju je osigurano dovoljno vremena za obustavu primjene transakcijskih zapisnika na sekundarni poslužitelj kako bi se sačuvali podaci od brisanja i na njemu.

[11]https://docs.microsoft.com/en-us/sql/database-engine/log-shipping/about-log-shipping-sql-server?view=sql-server-2017; 7.10.2019.

3.2. *Database mirroring* tehnologija

Database mirroring je jedna od tehnologija koja služi za osiguravanje visoke dostupnosti baza podataka *SQL Servera*. Radi na principu postojanja dvije kopije jedne baze koje se nalaze na različitim instancama *SQL Servera*.[12] Te se instance mogu nalaziti na istoj lokaciji ili, što je uobičajenije, na različitim lokacijama. Uspostavom *database mirrora* između dvije baze se inicira *database mirroring* sesija između te dvije instance.[13] *Database mirroring* je implementiran na razini baze, ali osim *principal* i *mirror* baze, ipak razlikujemo i *principal* i *mirror* poslužitelj. Iako se radi o zastarjeloj tehnologiji te će biti uklonjena iz budućih verzija *SQL Servera*, i dalje se koristi u poslovnom svijetu zbog svojih prednosti, npr. podržana je u *SQL Server Standard* ediciji koja je znatno jeftinija od *Enterprise* edicije pa se tvrtke radije opredjeljuju za ostanak na postojećoj tehnologiji zbog znatno manjeg ulaganja u licence.

Kako bi se objasnilo na kojem principu radi *database mirroring*, definirat će se osnovni pojmovi koji se koriste u ovoj tehnologiji.

***Principal* baza** je baza za čitanje i pisanje čiji se transakcijski zapisnici primjenjuju na *mirror* kopiji baze.[14]

***Mirror* baza** je kopija baze koja uobičajeno sinkronizirana s primarnom bazom i nalazi se na *mirror* poslužitelju.[15]

***Principal* poslužitelj** je onaj poslužitelj koji poslužuje bazu klijentu i sudjeluje kao partner u *database mirror* sesiji.[16]

***Mirror* poslužitelj** je onaj poslužitelj na kojem se nalazi *mirror* baza i koji djeluje kao *hot* ili *warm standby* poslužitelj.[17] Ako je *database mirror* sesija sinkronizirana, *mirror* poslužitelj predstavlja *hot standby* poslužitelj bez mogućnosti gubitka podatka, a ako nije, onda predstavlja *warm standby* poslužitelj s mogućim gubitkom podatka.

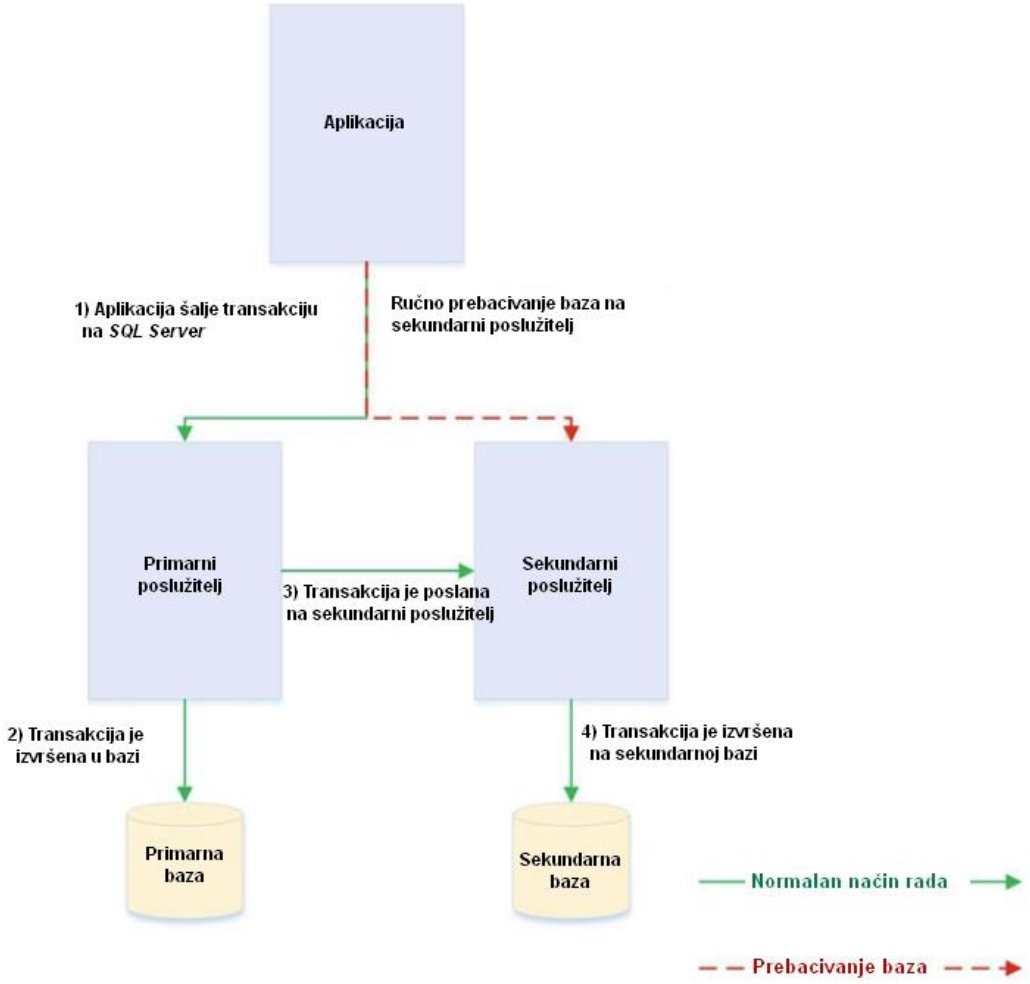
[12]https://docs.microsoft.com/en-us/sql/database-engine/database-mirroring/database-mirroring-sql-server?view=sql-server-2017; 9.10.2019.
[13] Ibidem; 9.10.2019.
[14] Ibidem; 20.11.2019.
[15] Ibidem; 20.11.2019.
[16] Ibidem; 20.11.2019.
[17] Ibidem; 20.11.2019.

Database mirroring može raditi u nekoliko različitih načina rada:

- *high-performance* način rada
- *high-safety* način rada
- *high-safety* način rada s automatskim prebacivanjem baza.

High-performance način rada

Ovaj način rada se odvija asinkrono te se transakcije kopiraju na *mirror* poslužitelj što je prije moguće i tamo se primjenjuju, ali se transakcije ne izvršavaju istovremeno na primarnom i sekundarnom poslužitelju, tj. primarni poslužitelj izvršava transakciju bez da pričeka sekundarni poslužitelj da dobije potvrdu da je zapis zapisan na njegov disk (Slika 3.2).



Slika 3.2 *Database mirroring* u *high-performance* načinu rada[18]

[18] SQL Server AlwaysOn Revealed, 2nd Edition; P. A. Carter; str.16.

Primarni poslužitelj daje potvrdu klijentu da je transakcija izvršena te ne čeka da bude izvršena i na sekundarnom poslužitelju. Zbog toga je u ovom načinu rada moguć gubitak podatka u slučaju gubitka primarnog poslužitelja. Stoga se *database mirroring* koji radi u asinkronom načinu rada smatra rješenjem za oporavak od katastrofe, dok se *database mirroring* u sinkronom načinu rada smatra rješenjem za visoku dostupnost.

High-safety način rada

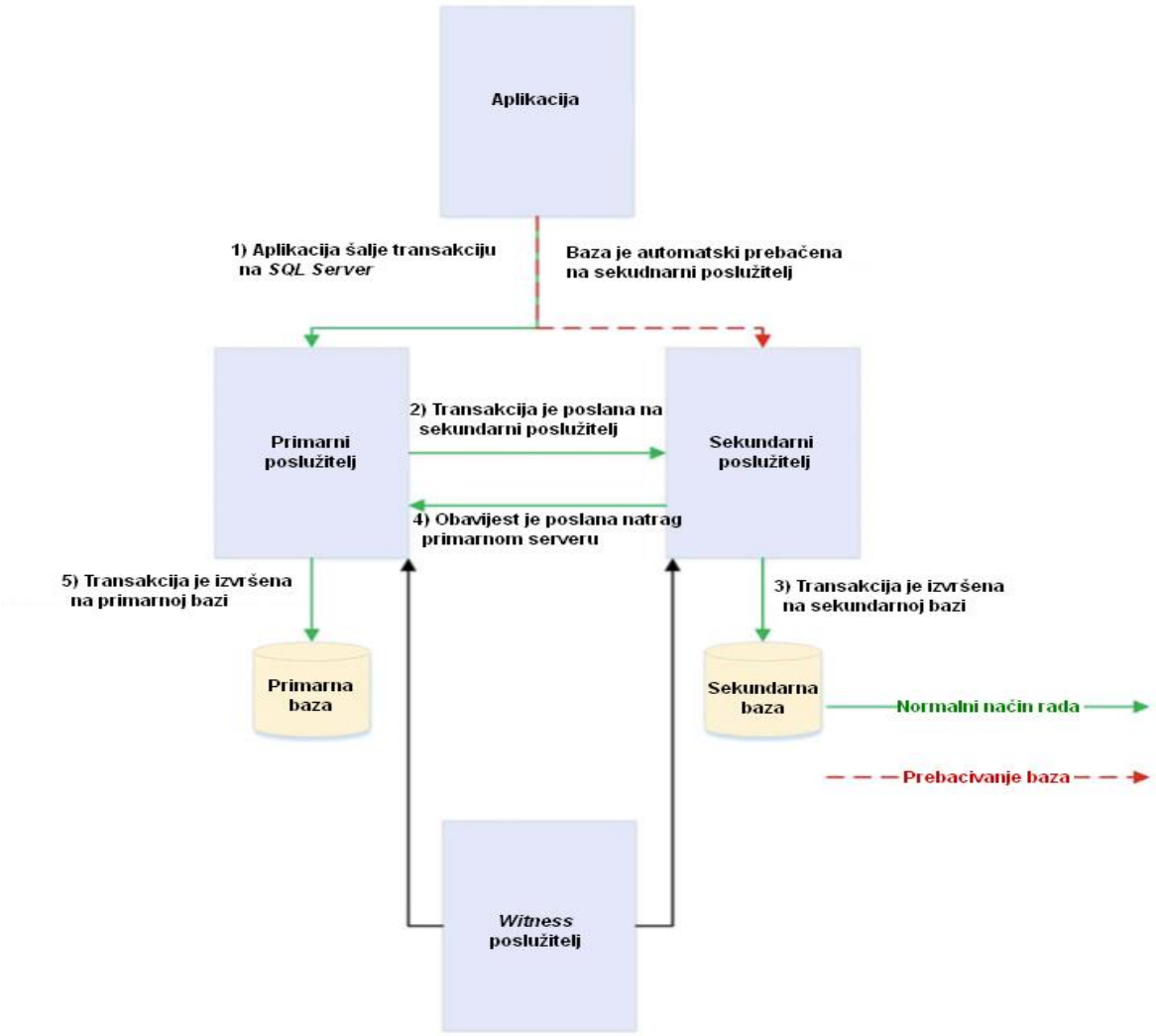
U ovom načinu rada transakcije se izvršavaju na sekundarnom poslužitelju koristeći sinkronu metodu, što znači da se najprije izvršavaju na sekundarnom poslužitelju, a tek onda na primarnom. Time je osigurano da niti jedna transakcija neće biti izvršena na primarnom poslužitelju prije no što dobijemo potvrdu od sekundarnog poslužitelja da je transakcija izvršena na njemu. Ovaj način rada ne podržava automatsko prebacivanje baza na sekundarni poslužitelj (engl. *automatic failover*) te je stoga pogodan samo za planirano prebacivanje baza na sekundarni poslužitelj. U tom slučaju se mora pokrenuti ručno prebacivanje baza na sekundarni poslužitelj koristeći korisničko sučelje na SQL poslužitelju ili izvršenjem naredbi za prebacivanje baze na sekundarni poslužitelj (Kôd 3.1).

```
ALTER DATABASE [DatabaseName] SET PARTNER FAILOVER
```

Kôd 3.1 Prikaz kôda za prebacivanje baze na sekundarni poslužitelj

High-safety način rada s automatskim prebacivanjem baza

Ovaj način rada podrazumijeva postojanje trećeg poslužitelja, kako bi postojao kvorum (engl. *quorum*) te koji služi kao svjedok (engl. *witness server*) koji omogućuje automatsko prebacivanje baza u slučaju gubitka *principal* poslužitelja. Treći poslužitelj ne poslužuje baze podataka, već služi samo kao arbitar u slučaju da *principal* i *sekundarni* poslužitelj izgube povezanost. To znači da svaki od poslužitelja koji čine kvorum vrijedi jedan glas, a za dostupnost baza potrebna su najmanje dva glasa. Stoga *principal* poslužitelj mora biti povezan na najmanje jednu od preostalih instanci kako bi se očuvao kvorum (na *mirror* poslužitelj ili *witness* poslužitelj). U protivnom, baze na njemu postaju nedostupne. Ako su međusobno mrežno dostupni *mirror* poslužitelj i *witness* poslužitelj, a nije dostupan *principal* poslužitelj, u ovom načinu rada doći će do automatskog prebacivanja baza na *mirror* poslužitelj, odnosno *mirror* poslužitelj će preuzeti ulogu *principal* poslužitelja te će na njemu baze postati dostupne kao primarne baze podatka. U ovom načinu rada transakcije se izvršavaju na isti način kao i *high-safety* načinu rada, a jedina razlika je u tome što ovdje imamo automatsko prebacivanje baza uz pomoć *witness* poslužitelja (Slika 3.3).



Slika 3.3 *Database mirroring* u *high safety with Automatic Failover* načinu rada[19]

Najpouzdaniji način rada je svakako *High Safety with Automatic Failover*, koji u potpunosti osigurava od gubitka podatka, ali je to ujedno i najskuplje rješenje jer je potrebno osigurati dodatni *witness* poslužitelj.

Nedostaci i ograničenja *Database mirroringa* su:

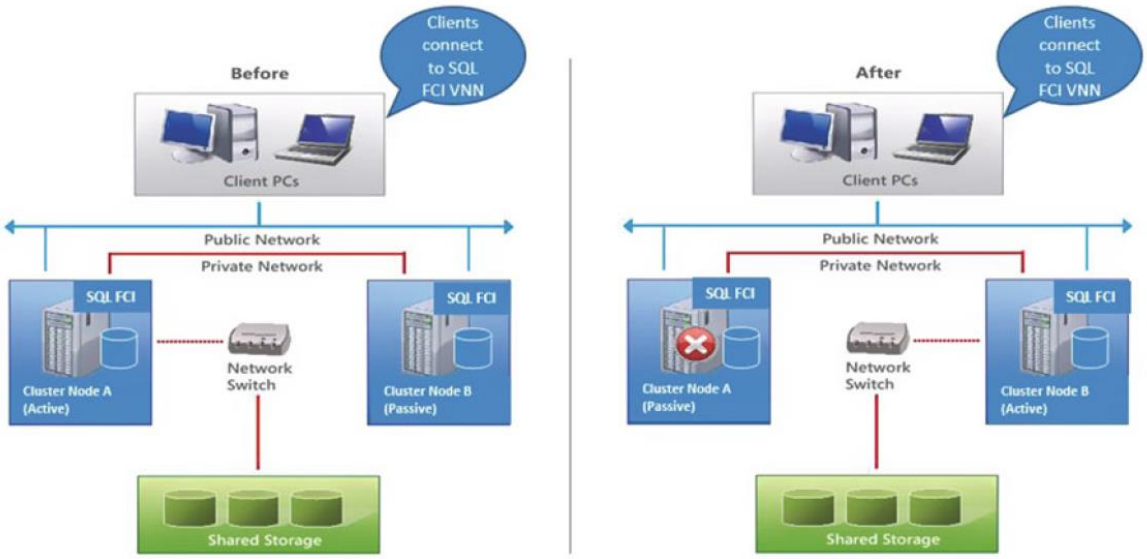
- Nema mogućnosti uspostave više sekundarnih kopija.
- Djeluje na razini baze te nije moguće grupno prebacivanje baza na *mirror* poslužitelj.
- Iz perspektive troškova, *mirror* poslužitelj radi većinu vremena neiskorišten jer se ne može pristupiti njegovim bazama.

Većina navedenih nedostataka je riješena s uvođenjem *AlwaysOn* tehnologije te se zato *database mirroring* i smatra zastarjelom tehnologijom.

[19] Ibidem; str. 17.

3.3. Failover Cluster Instances (FCI) tehnologija

Failover Cluster Instance (skraćeno FCI) predstavlja jednu instancu *SQL Servera* koja je instalirana kroz više *Windows Server Failover Clustering* (skraćeno WSFC) čvorova (engl. *node*).[20] Točnije, *SQL Server* instaliran je na svakom od čvorova u klasteru, ali je pokrenut samo na jednom, koji ujedno predstavlja aktivni čvor. Podaci *SQL Servera* nalaze se na dijeljenom sustavu za pohranu podatka (engl. *shared storage*) te se drugi čvor, nakon prebacivanja instance na njega, spaja na dijeljeni sustav za pohranu podatka te prezentira podatke krajnjem korisniku (Slika 3.4).



Slika 3.4 Dijagram *SQL failover clustera*[21]

Ako se aktivni čvor pokvari, tada se na njemu zaustavi *SQL Server* i od njega se odvoji zajednički sustav za pohranu podataka te se poveže na neki od preostalih čvorova u klasteru. Na tom se čvoru pokrene *SQL Server* te on postaje aktivni čvor.

Windows Server Failover Cluster predstavlja grupu neovisnih poslužitelja koji rade zajedno na povećanju dostupnosti i servisa.[22] Ukupno 64 poslužitelja može biti povezano u *windows cluster*, osiguravajući tako visoku dostupnost te redundanciju.

[20] Pro SQL Server Always On Availability Groups; Uttam Parui, Vivek Sanil; str. 10.
[21] Ibidem; str. 10.
[22]https://docs.microsoft.com/en-us/sql/sql-server/failover-clusters/windows/windows-server-failover-clustering-wsfc-with-sql-server?view=sql-server-2017; 5.12.2019.

Klijent na mreži vidi FCI kao jednu instancu *SQL Servera* te se spaja koristeći virtualno mrežno ime (engl. *virtual network name*, skraćeno VNN). Tako klijent i nakon što dođe do prebacivanja *SQL* instance na drugi čvor i dalje koristi isto virtualno mrežno ime za spajanje na aktivnu *SQL* instancu. Stoga ovdje, za razliku od *database mirroringa* ili *log shippinga*, nema dupliciranja podataka jer se svi podaci nalaze na istom dijeljenom sustavu za pohranu podataka. To je ujedno i jedinstvena točka kvara (engl. *single point of failure*), što ujedno predstavlja nedostatak ovakvog sustava jer, ako dođe do kvara ili uništenja dijeljenog sustava za pohranu podataka, gubimo sve podatke te se rješenje mora potražiti u dodatnom sustavu za oporavak od katastrofe.[23]

3.4. *AlwaysOn* tehnologija

AlwaysOn predstavlja kolekciju funkcionalnosti za visoku dostupnost i oporavak od katastrofe s ciljem da se *recovery point objective* (RPO) i *recovery time objective* (RTO) minimiziraju ispod vremena koje se može postići prethodno opisanim rješenjima.[24] *SQL Server 2016* donio je brojne novosti u pogledu otklanjanja ograničenja koja su inicijalno postojala unutar *AlwaysOn* rješenja te u novoj, osvježenoj inačici sada uključuje:

- *AlwaysOn Availability Groups*
- *AlwaysOn Failover Cluster Instances.*

3.4.1. *AlwaysOn Failover Cluster Instances* tehnologija

Budući da je u poglavlju 3.3 (Failover Cluster Instances (FCI)) već opisan princip rada ove tehnologije, u ovom poglavlju će biti više riječi o opisu razlika i poboljšanja koje je donio *SQL Server 2016* u odnosu na raniji *Failover Cluster Instances*, koji je do tada postojao kao zasebna funkcionalnost *SQL Servera*.

AlwaysOn Failover Cluster Instances donosi sljedeća poboljšanja:

- mogućnost implementacije klastera između više podmreža korištenjem *Storage Replica* tehnologije
- mogućnost implementacije klastera između više različitih domena.

[23] Pro SQL Server Internals, Second Edition; Dmitri Korotkevitch; str. 671.
[24] Pro SQL Server Always On Availability Groups; Uttam Parui, Vivek Sanil; str. 14.

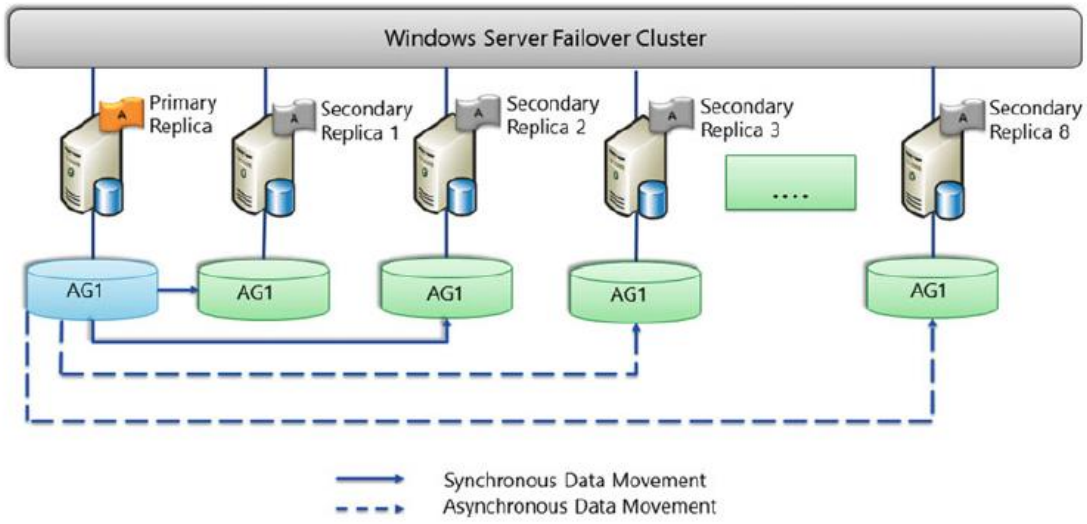
Sve do izlaska *SQL Servera 2016* te *Windows Servera 2016* nije bila moguća implementacija klastera između udaljenih lokacija jer je podrazumijevala replikaciju umreženih spremišta podatka (engl. *Storage Area Network*, skraćeno SAN), koja podrazumijeva da je klaster konfiguriran unutar iste lokacije. Ovi nedostatci su otklonjeni s izlaskom *Windows Servera 2016* koji je omogućio da *Storage Replica* osigura *geo-cluster* rješenje bez SAN replikacije. *Storage Replica* radi na principu sinkronizacije na razini blokova podataka (engl. *block level data synchronization*) te može raditi u sinkronom ili asinkronom načinu rada.[25]

3.4.2. *AlwaysOn* grupe visoke dostupnosti – topologija i načini rada

***AlwaysOn* grupe visoke dostupnosti** (engl. *AlwaysOn Availability Groups,* skraćeno AOAG) zamijenile su *database mirroring*, a predstavljaju spoj *database mirroringa* i *clustering tehnologija*[26]. *SQL Server* instaliran je kao zasebna instalacija na svaki od čvorova u klasteru, dok je na samom klasteru instalirana aplikacija *Availability Group Listener* (skraćeno *AG listener*), koja je svjesna klastera (engl. *cluster-aware application*), a služi preusmjeravanju prometa prema ispravnom čvoru klastera.

AlwaysOn Availability Groups tehnologija radi na principu sažimanja (engl. *compression*) i slanje blokova transakcijskih zapisa (engl. *transaction log blocks*) na drugi čvor, odnosno čvorove u klasteru, sinkrono ili asinkrono, ovisno o konfiguraciji. Za razliku od *database mirroringa*, koji ne podržava više sekundarnih kopija, AOAG podržavaju do 8 sekundarnih kopija uključujući 2 sinkrone (Slika 3.5).

[25] SQL Server AlwaysOn Revealed, 2nd Edition; P. A. Carter; str.; str. 9.
[26] Ibidem; str. 18.



Slika 3.5 Primjer *AlwaysOn* rješenja s 2 sinkrone i 6 asinkronih sekundarnih replika[27]

Grupe visoke dostupnosti *(engl. Availability Group*, skraćeno AG) predstavljaju spremnik (engl. *container*) korisničkih baza podataka koje se prebacuju na sekundarni poslužitelj zajednički kao grupa.[28] Svi objekti unutar *SQL Servera* koji postoje izvan baza podataka koje se nalaze u grupi (korisnici, zadaci, povezani poslužitelji…), neće se prebaciti zajedno s AG.

Replike dostupnosti (engl. *Availability replicas*) predstavljaju set od dva ili više partnera koji mogu sudjelovati u prebacivanju visoko dostupnih grupa na sekundarni poslužitelj.[29] Svaka replika drži kopiju visoko dostupnih baza podataka te se mora nalaziti na različitom čvoru istog *Windows Server Failover Clustera* (skraćeno WSFC).

Razlikujemo dvije vrste visoko dostupnih replika:

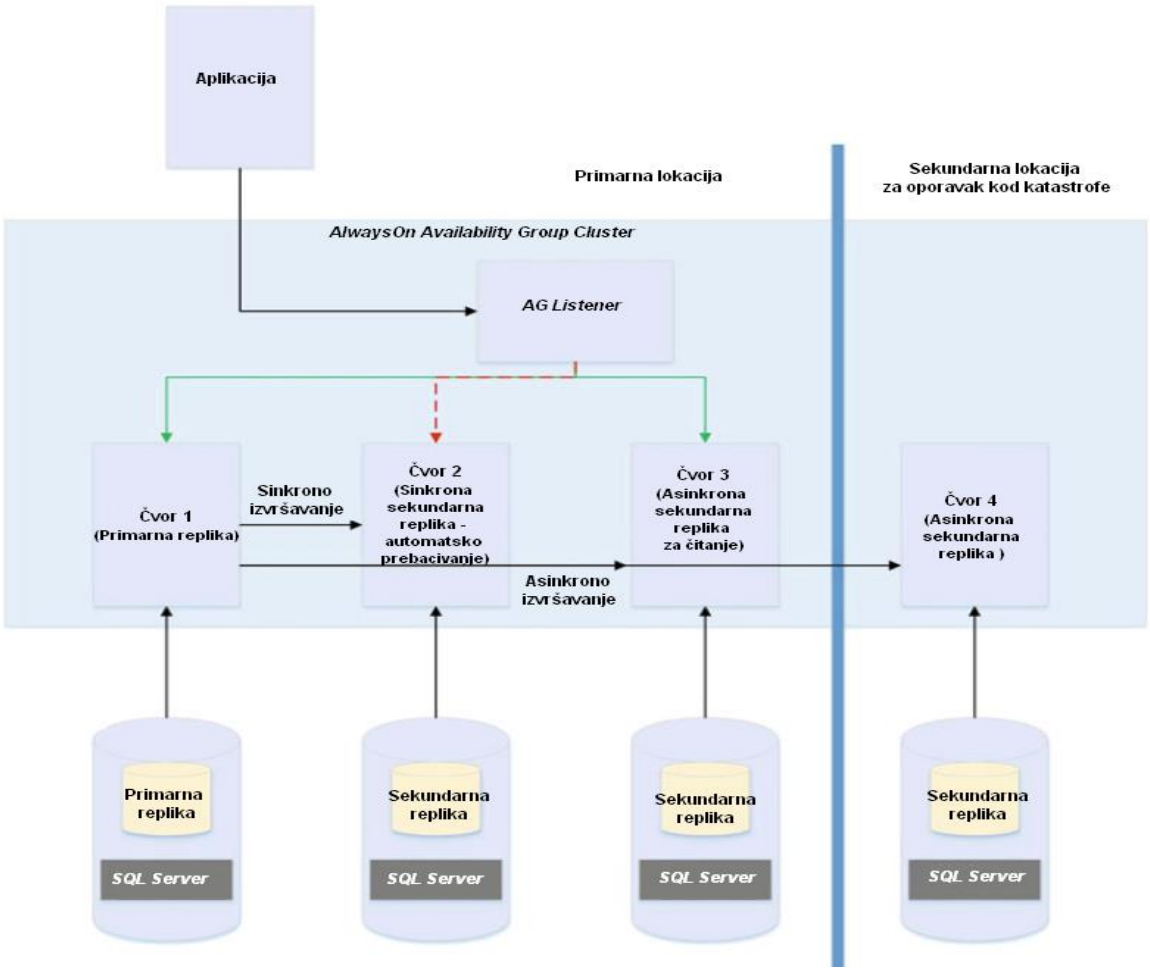
- *primarna replika*
- *sekundarna replika*.

Primarna replika je replika na kojoj se nalaze primarne baze i čini ih dostupnima klijentima za čitanje i pisanje. [30]

[27] Ibidem; str. 19.
[28] Pro SQL Server Always On Availability Groups; Uttam Parui, Vivek Sanil; str. 20.
[29] https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/overview-of-always-on-availability-groups-sql-server?view=sql-server-2017; 5.12.2019.
[30] Pro SQL Server Always On Availability Groups; Uttam Parui, Vivek Sanil; str. 20.

Sekundarna replika je replika koja održava sekundarnu kopiju svake primarne baze i služi kao potencijalna meta za prebacivanje baza zajedno s visoko dostupnom grupom.[31] Sekundarna replika također može biti aktivna jer podržava *read-only* pristup sekundarnim bazama te može poslužiti kao sustav izvještavanja.

AlwaysOn tehnologija može istovremeno poslužiti kao rješenje za visoku dostupnost i za oporavak od katastrofe u slučaju konfiguracije s više replika raspoređenih na primarnoj i sekundarnoj lokaciji (engl. DR Site) koje rade u različitim načinima rada (Slika 3.6).



Slika 3.6 *AlwasyOn Availability Group* topologija[32]

Kao što je vidljivo na slici (Slika 3.6), u ovom primjeru imamo 4 čvora (engl. *node*) u klasteru te na svakom od njih imamo po jednu instancu *SQL Servera* koja drži replike baze. Na prvom čvoru nalazi se primarna replika baze, dok drugi čvor služi za automatsko prebacivanje baza (engl. *automatic-failover*). Treći čvor predstavlja asinkronu repliku

[31] Ibidem; str. 20.
[32] SQL Server AlwaysOn Revealed, 2nd Edition; P. A. Carter; str. 21.

primarne baze koja radi u *read-only* načinu te stoga služi kao *reporting* poslužitelj koji rasterećuje primarnu repliku baze, budući da *read-only* upiti dolaze samo na njega. Četvrti poslužitelj, s replikom u asinkronom načinu rada, nalazi se na udaljenoj lokaciji koja se koristi samo u slučaju oporavka od katastrofe (engl. *disaster recovery site*).

Sinkroni i asinkroni način

Primarna replika šalje blokove transakcijskih zapisa svake primarne baze na sve sekundarne replike. Ovisno o tome čeka li primarna replika izvršavanje transakcije na bazi dok sekundarna replika ne zapiše transakcijski zapis na disk, razlikujemo sinkroni (engl. *Synchronous-commit mode*) i asinkroni način izvršavanja (engl. *Asynchronous-commit mode*).

U sinkronom načinu rada primarna replika čeka da klijentu pošalje potvrdu da je transakcija izvršena sve dok sekundarna replika ne zapiše transakcijski zapis (engl. *transaction log*) na disk[33]. Ovaj način rada jamči da neće doći do gubitka podataka te je RTO jednak 0, ali zbog toga ima utjecaj na performanse i povećava latenciju izvršenja transakcija (engl. *transaction latency*).

U asinkronom načinu rada primarna replika ne čeka sekundarnu repliku da zapiše transakcijski zapis na disk, nego šalje potvrdu o transakciji klijentu čim blok transakcijskog zapisa postoji u primarnoj bazi.[34] Ovaj način rada ima znatno manje latencije pri izvršenju transakcija, ali ne garantira da neće doći do gubitka podataka.

Automatsko i ručno prebacivanje dostupnih grupa

Prebacivanje visoko dostupnih grupa (engl. *failover*) na sekundarni poslužitelj je proces tijekom kojeg sekundarna replika preuzima rolu primarne, oporavlja baze za korištenje te ih podiže *online* kao nove primarne baze.[35]

Automatsko prebacivanje (engl. *automatic failover*) dostupnih grupa pokreće se u slučaju postojanja problema na primarnoj replici. Pod ručnim prebacivanjem (engl. *manual failover*) dostupnih grupa podrazumijeva se prebacivanje na sekundarnu repliku inicirano od strane administratora baza podatka.

---

[33] Pro SQL Server Always On Availability Groups; Uttam Parui, Vivek Sanil; str. 22.
[34] Ibidem; str. 22.
[35]https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/failover-and-failover-modes-always-on-availability-groups?view=sql-server-ver15; 9.12.2019.

4. Implementacija *AlwaysOn* grupa visoke dostupnosti

4.1. Preduvjeti za instalaciju

Za implementiranje *AlwaysOn grupe visoke dostupnosti* najprije je potrebno ispuniti preduvjete u pogledu verzije operativnog sustava, verzije i edicije *SQL Servera* te same baze podatka, a zatim i kreirati *Windows Server Failover Cluster* između čvorova koji sudjeluju u *AlwaysOn* grupama.

Preduvjeti na razini operativnog sustava

Što se tiče operativnih sustava koji su podržani za implementaciju *AlwaysOn* sustava, potreban je minimalno *Windows Server 2008* jer podržava *Windows Server Failover Clustering* funkcionalnost. Sve do *Windows Servera 2012 R2* klaster je mogao biti kreiran između čvorova unutar iste domene, a od *Windows Servera 2016* moguće je implementirati klaster čak i bez ovisnosti o domeni.

Preduvjeti na razini SQL Servera

U pogledu verzije *SQL Servera,* implementacija *AlwaysOn* grupa moguća je na verzijama od *SQL Servera 2012* na više.

Da bi se unutar *SQL Servera* implementirale *AlwaysOn* grupe visoke dostupnosti, potrebno je ispuniti preduvjete na razini svake *SQL Server* instance koja sudjeluje u grupi visoke dostupnosti:

- Poslužitelj na kojem je instalirana instanca *SQL Servera* mora biti *Windows Server Failover Clustering* čvor.
- Svaka instanca *SQL Servera* na kojoj se nalazi replika visoko dostupne grupe mora se nalaziti na odvojenom čvoru klastera.
- *AlwaysOn* grupe moguće je konfigurirati samo na *Enterprise* ediciji *SQL Servera*.
- Sve instance *SQL Server* koje sudjeluju u istoj visoko dostupnoj grupi moraju imati isti *SQL Server collation* (pravila sortiranja te osjetljivost na velika i mala slova i naglaske pri usporedbi).
- Svaka *SQL Server* instanca koja drži repliku mora imati omogućenu *AlwasyOn Availability Groups* funkcionalnost (engl. *feature*).

Preduvjeti na razini baze podataka

Kako bi baza podatka mogla sudjelovati u grupi visoke dostupnosti, mora ispunjavati sljedeće preduvjete:[36]

- Mora se raditi o korisničkoj bazi podatka, ne može biti sistemska.
- Mora imati uključenu opciju za korištenje od strane više korisnika (engl. *multiuser database*).
- Mora imati uključenu opciju automatskog zatvaranja (engl. *auto close*).
- *Recovery model* baze mora biti postavljen na *full*.
- Baza mora imati napravljenu barem jednu potpunu sigurnosnu kopiju (engl. *full backup*).
- Ne smije pripadati u postojeću grupu visoke dostupnosti.
- Ne smije sudjelovati u *database mirroringu*.

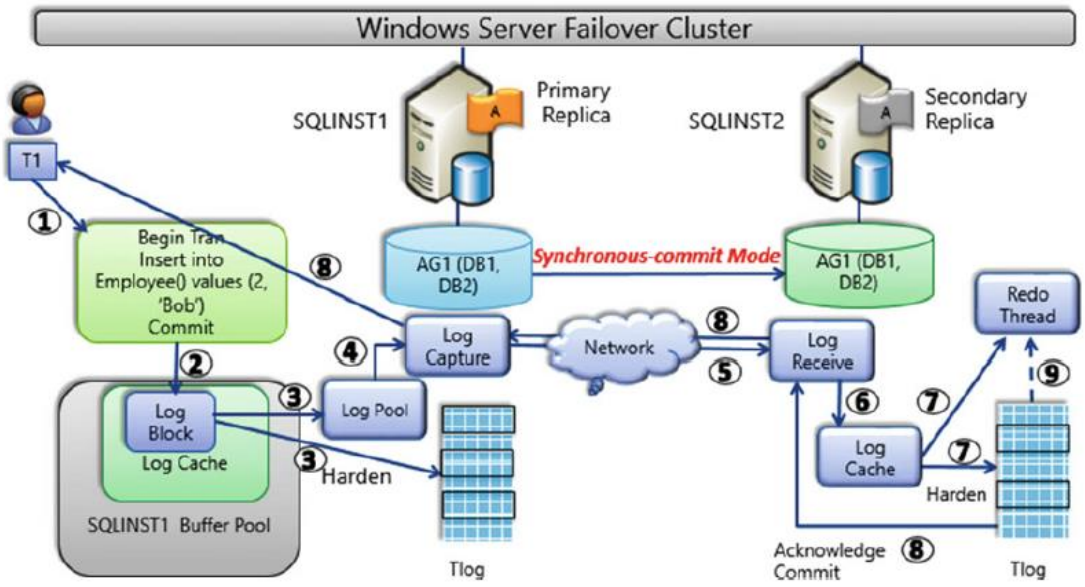
4.2. Princip rada i funkcionalnosti

Kako bi se razjasnio princip rada *AlwasyOn* visoko dostupnih grupa, u ovom će poglavlju biti opisano na koji način radi repliciranje blokova transakcijskih zapisa (engl. *transaction log blocks*) s primarne replike na sekundarnu u sinkronom i asinkronom načinu rada. Nakon toga će biti lakše odlučiti se u kojem slučaju je bolje odabrati sinkroni način, a u kojem asinkroni. Razumjet će se zašto je sinkroni način izvršenja transakcija sporiji, ali osigurava veću dostupnost te zašto je asinkroni način brži, ali nepouzdaniji s aspekta dostupnosti i očuvanja podataka.

Koraci sinkronog načina izvršenja podataka (engl. *Synchronous-Commit Mode*)

Kako bi se podaci sinkrono prenosili s primarne na sekundarnu repliku obje moraju biti konfigurirane za sinkroni način izvršenja transakcija. Sinkronizacija podatka između primarne i sekundarne replike u sinkronom načinu rada odvija se na način prikazan na slici (Slika 4.1).

[36] https://www.sqlrx.com/prerequisites-for-installing-sql-server-alwayson/; 11.12.2019.



Slika 4.1 Sinkronizacija podataka u sinkronom načinu rada[37]

Sinkronizacija podataka u sinkronom načinu rada odvija se u sljedećim koracima[38]:

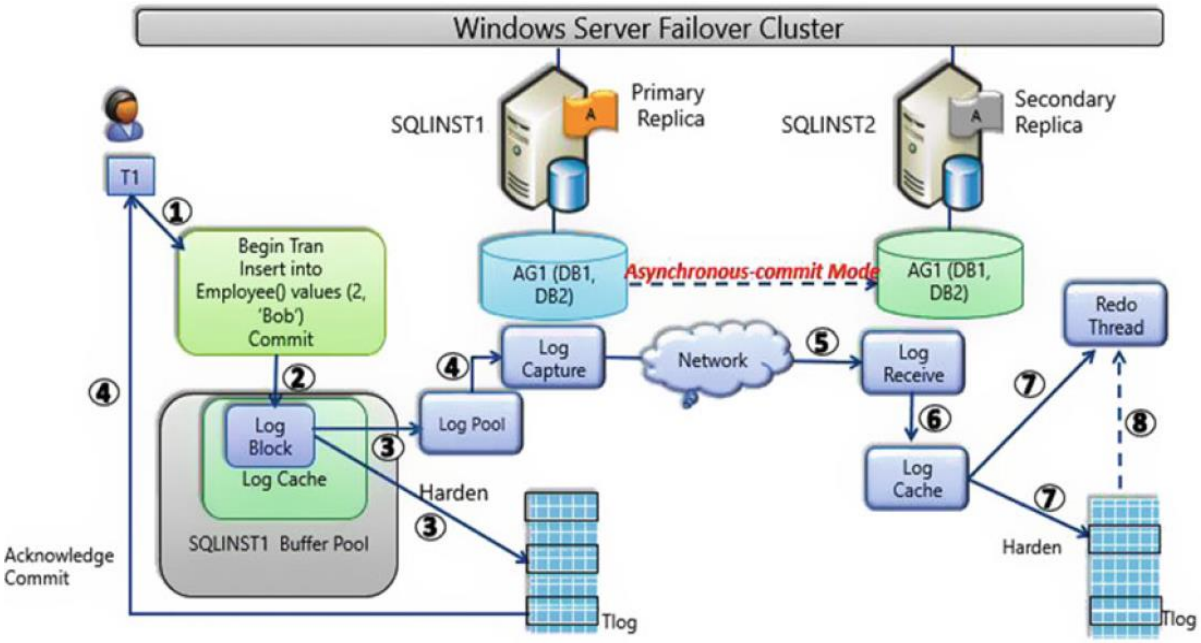
1. Klijent upućuje transakciju prema bazi na primarnoj replici, koja sudjeluje u visoko dostupnoj grupi.
2. Primarna replika generira blok transakcijskog zapisa (engl. *transaction log block)* te zatim primarna i sekundarna replika dogovaraju koji je slijedni broj zapisa (engl. *log sequence number*) početna točka za prijenos. Na primarnoj replici tada se u predmemoriju zapisa (engl. *log cache*) zapisuje taj blok zapisa.
3. Kada blok zapisa postane pun ili kada se na primarnoj replici izvrši transakcija, tada se blok zapisa iz predmemorije zapisa prepiše u zapisničku datoteku (engl. *log file*), kako bi postao trajno zapisan. Paralelno sa zapisivanjem u zapisničku datoteku sprema se i kopija u bazen zapisa (engl. *log pool*).
4. Tada blokove loga iz bazena zapisa (engl. *log pool*) čita dretva za hvatanje zapisa (engl. *log capture thread*) te ih šalje na sekundarnu repliku.
5. Na sekundarnoj replici nalazi se dretva za primanje zapisa (engl. *log receive thread*) koja prima blokove zapisa s mreže.

[37] Pro SQL Server Always On Availability Groups; Uttam Parui, Vivek Sanil; str. 36.
[38] Ibidem; str. 36.

6. Dretva za primanje zapisa ujedno i zapisuje blokove zapisa u predmemoriju zapisa na sekundarnoj replici.
7. Na svakoj bazi sekundarne replike nalazi se dretva pod nazivom *redo thread*, koja čita blokove zapisa iz predmemorije zapisa i primjenjuje promjene na podatkovne stranice i stranice indeksa u bazi na sekundarnoj replici.
8. Sekundarna replika sada šalje potvrdu o izvršenju transakcije primarnoj replici, kako bi mogla javiti aplikaciji da je transakcija izvršena. S obzirom na to da je transakcija zapisana na sekundarnoj replici, imamo garanciju da neće doći do gubitka podataka u slučaju prebacivanja baza na sekundarni poslužitelj.
9. *Redo thread* izvršava se neovisno o pristizanju blokova zapisa, tako da se može dogoditi da se izvršio nekoliko minuta poslije pa da taj blok zapisa ne postoji u predmemoriji. U tom slučaju će *redo thread* pokupiti te blokove zapisa iz zapisničke datoteke sa diska.

Koraci asinkronog načina izvršenja podataka (engl. *Asynchronous-Commit Mode*)

Asinkroni način sinkronizacije podatka između primarne i sekundarne replike može biti konfiguriran s namjerom, ali isto tako u asinkroni način rada ponekad primarna replika ulazi kada se za to ostvare preduvjeti. Primarna i sekundarna replika provjeravaju međusobnu dostupnost, slanjem ping signala. Ako primarna replika ne dobije odgovor na *ping* zahtjev unutar vremena određenog postavkom *session-timeout period* (zadano 10 sekundi), primarna replika se prebacuje u asinkroni način rada za tu sekundarnu repliku. Kada sekundarna replika ponovo postane dostupna, vraća se automatski u sinkroni način rada. Sinkronizacija podatka između primarne i sekundarne replike u asinkronom načinu rada odvija se na način prikazan na slici (Slika 4.2).



Slika 4.2 Sinkronizacija podataka u asinkronom načinu rada[39]

Koraci sinkronizacije podataka u asinkronom načinu rada su sljedeći[40]:

1. Klijent upućuje transakciju prema bazi na primarnoj replici, koja sudjeluje u visoko dostupnoj grupi.
2. Primarna replika generira blok transakcijskog zapisa te primarna i sekundarna replika dogovaraju koji je slijedni broj zapisa početne točka za prijenos. Na primarnoj replici se tada u predmemoriju zapisa zapisuje taj blok zapisa.
3. Kada blok zapisa postane pun ili kada se na primarnoj replici izvrši transakcija, tada se blok zapisa iz predmemorije zapisa isprazni u zapisničku datoteku, kako bi postao trajno zapisan. Paralelno sa zapisivanjem, u zapisničku datoteku se sprema i kopija u bazen zapisa.
4. Primarna replika šalje potvrdu aplikaciji da je transakcija izvršena kada je zapisivanje transakcijskog zapisa u zapisničku datoteku na primarnoj replici završeno. Paralelno sa zapisivanjem u zapisničku datoteku, sprema se i kopija u bazen zapisa. Tada blokove zapisa iz bazena zapisa čita dretva za hvatanje zapisa te ih šalje na sekundarnu repliku. Ako ima više sekundarnih replika, za svaku repliku postoji po jedna dretva za hvatanje zapisa. Sadržaj zapisa dodatno se kompresira i kriptira prije slanja na sekundarnu repliku.

[39] Ibidem; str. 38
[40] Ibidem; str. 38.

5. Na sekundarnoj replici nalazi se dretva za primanje zapisa koja prima blokove zapisa s mreže.
6. Dretva za primanje zapisa ujedno i zapisuje blokove zapisa u predmemoriju zapisa na sekundarnoj replici.
7. Na sekundarnoj replici nalazi se dretva pod nazivom *redo thread*, koja čita blokove zapisa iz predmemorije zapisa i primjenjuje promjene na podatkovne stranice i stranice indeksa u bazi na sekundarnoj replici.
8. *Redo thread* izvršava se neovisno o pristizanju blokova zapisa, tako da se može dogoditi da se izvršio nekoliko minuta poslije pa da taj blok ne postoji zapisan u predmemoriji. U tom slučaju će *redo thread* dohvatiti te blokove zapisa iz zapisničke datoteke sa diska.

4.3. Uspostavljanje i konfiguracija

U ovom poglavlju opisat će se koraci implementacije *AlwaysOn* dostupnih grupa na virtualnim poslužiteljima u tvrtki u kojoj radi autor rada, implementiranim isključivo u svrhu pisanja završnog rada i testiranja *AlwaysOn* tehnologije. Kao glavni poslužitelji (engl. *host*) pod nazivom TESTNODE01, TESTNODE02 i TESTNODE03 iskorišteni su fizički poslužitelji marke Lenovo, model SR530. Navedeni fizički poslužitelji imaju sljedeća svojstva:

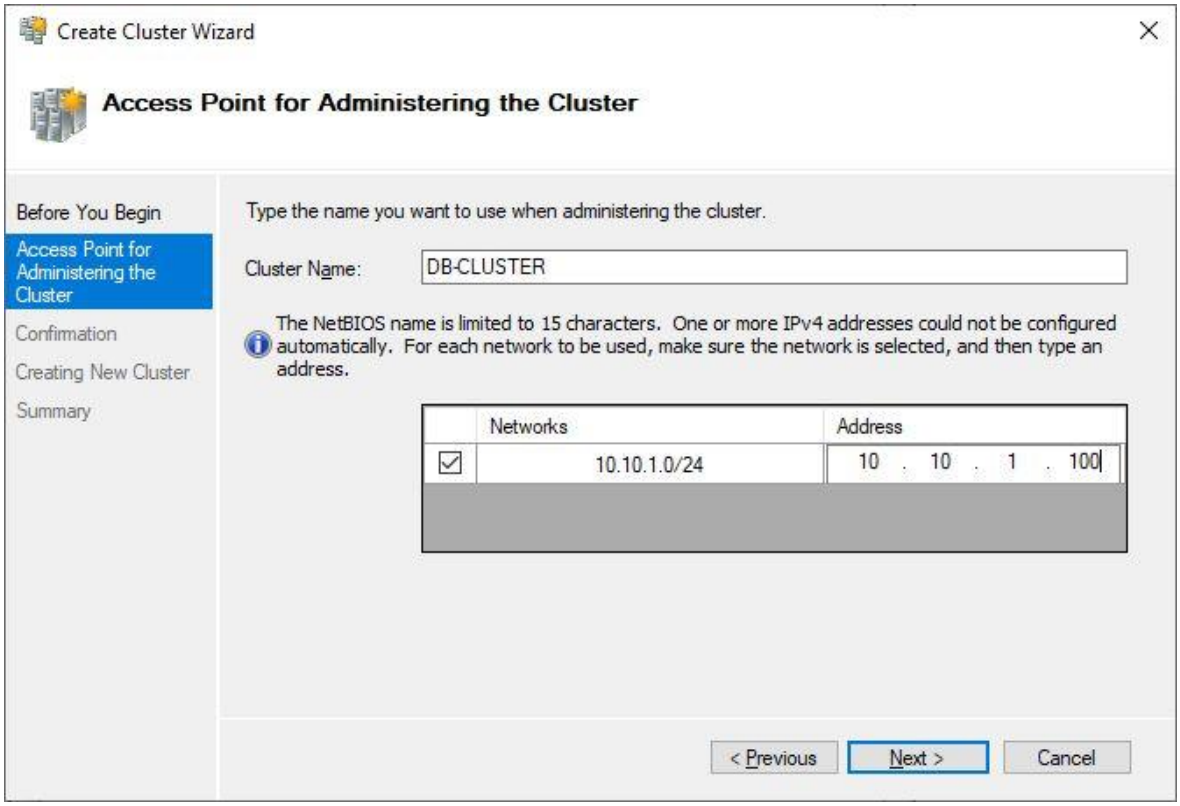
- radna memorija: 128 GB
- procesor: Intel Xeon Silver 4108 s 8 jezgri
- diskovni sustav: 1 TB SSD disk u RAID 1 polju + 1 TB SATA disk u RAID 1 polju.

Za potrebe ovog rada instalirani su virtualni poslužitelji unutar *Hyper-V* okruženja. Nazvani su sljedećim imenima: DC, DB01, DB02 i DB03. Svaki od DB poslužitelja stavljen je na zaseban, *host* poslužitelj, kako bi nakon dodavanja u *Windows Server Failover Cluster* bili svaki u zasebnom čvoru klastera. Na svim poslužiteljima instaliran je *Windows Server 2019* operativni sustav. Poslužitelj pod nazivom DC predstavlja domenski poslužitelj (engl. *Domain Controller*), na kojem je podignuta domena pod nazivom VGdomain.test. Nakon kreiranja domene, u nju su pridružena i preostala tri poslužitelja (DB01, DB02 i DB03), koji će sudjelovati kao replike u *AlwaysOn* grupama. Zbog fokusa ovog rada na osiguranje visoke dostupnosti te *AlwaysOn* tehnologiju, postupak kreiranja domene i dodavanja poslužitelja u domenu izdvojeni su u poseban dokument koji će biti priložen ovom radu.

Instalacija i konfiguracija *Windows Server Failover Clustera*

Kako bi se na instanci *SQL Servera* mogao uključiti *AlwaysOn*, najprije je potrebno unutar *Server Managera* dodati funkcionalnost *Failover Clustering*. Ovu radnju moramo ponoviti na svim poslužiteljima na kojima se nalaze instance *SQL Servera* koje će sudjelovati u *AlwaysOn* dostupnim grupama.

Sada se tek može pristupiti kreiranju klastera. Najprije treba validirati klaster unutar *Failover Cluster Managera*, tako da se u izborniku *Validate Configuration* odaberu svi poslužitelji koji će sudjelovati u klasteru te se, nakon što validacija uspješno završi, odabere naziv klastera te IP adresa koja će mu biti dodijeljena, kako je prikazano na slici (Slika 4.3).

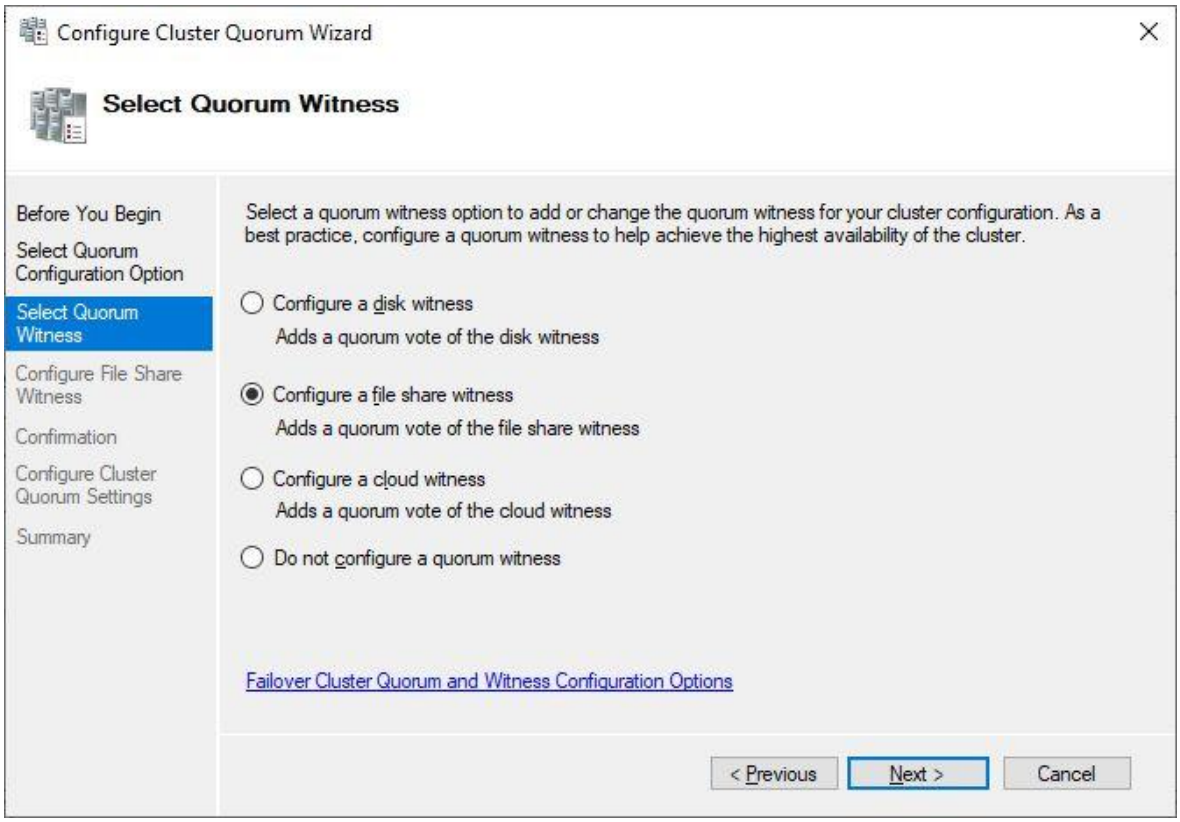


Slika 4.3 Odabir naziva klastera

Kvorum klastera i *file share witness*

Sada još treba konfigurirati kvorum klastera. Elemente klastera čine čvorovi klastera te u nekim konfiguracijama i *disk witness* ili *file share witness*. Ako dovoljan broj elemenata klastera nije dostupan, klaster gubi kvorum te se zaustavlja s radom. Klaster ima kvorum sve dok je više od pola elemenata koji ga čine dostupno. Kvorum je u klasteru potreban kako bi se izbjegao *split-brain scenario*. Radi se o situaciji kada nastanu problemi u komunikaciji između čvorova u klasteru te više replika istovremeno, smatrajući da je primarna replika,

pokušava kod sebe uspostaviti dostupnu grupu i stoga prihvaćaju i odgovara na upite.[41] Uvođenjem *file share witnessa* dobiva se još jedan glas u kvorumu te je na taj način spriječen ovakav scenarij. Za potrebe ovog rada odabran je *file share witness*, kao što je prikazano na slici (Slika 4.4).



Slika 4.4 Odabir *quorum witnessa*

File share witness zapravo je dijeljena datoteka koja ima jedan glas u kvorumu kao i preostali čvorovi koji sudjeluju u klasteru. Za klaster pod nazivom DB-CLUSTER odabran je *file share witness* naziva DB_Witness koji se nalazi na DC poslužitelju.

Omogućavanje *AlwaysOn* funkcionalnosti na *SQL* instanci

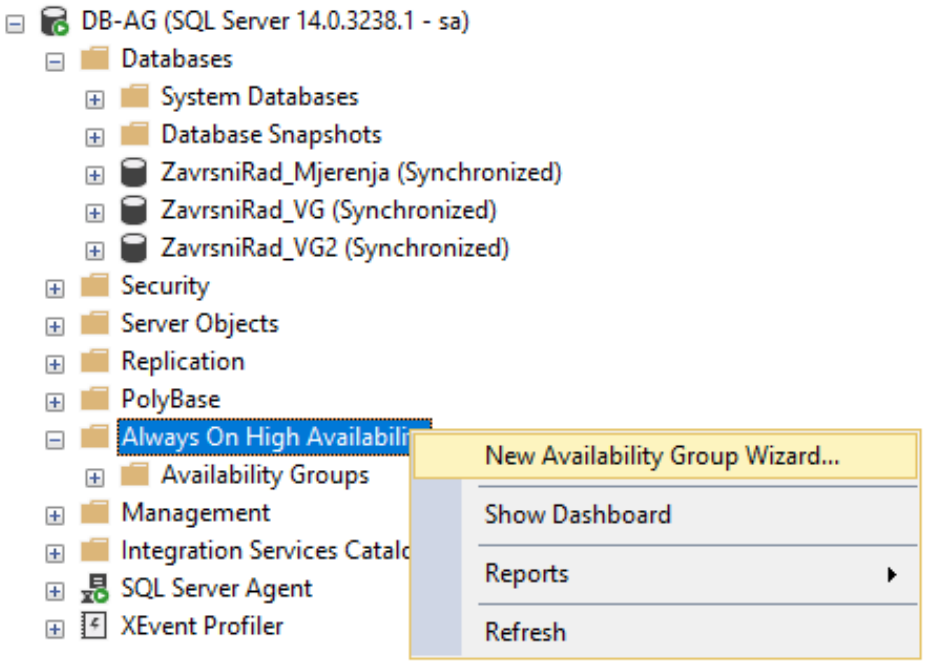
AlwaysOn funkcionalnost se uključuje tako da se unutar *SQL Server Configuration Manage*ra pod postavkama *SQL* servisa odabere izbornik *AlwaysOn High Availability* te tamo omogući *AlwasyOn* funkcionalnost i odabera naziv klastera (u ovom slučaju DB-CLASTER). Ovu radnju potrebno je ponoviti na svim instancama koje će sudjelovati u *AlwasyOn* dostupnim grupama (u ovom slučaju DB01, DB02 i DB03 instance).

[41] Pro SQL Server Always On Availability Groups; Uttam Parui, Vivek Sanil; str. 64.

Kreiranje visoko dostupnih grupa

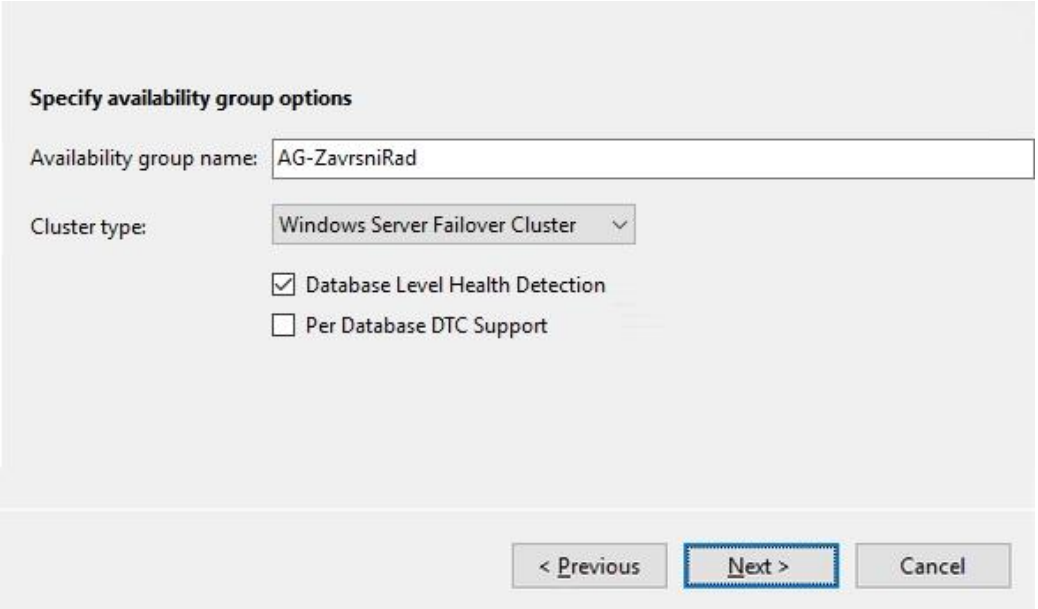
Dostupne grupe moguće je kreirati na više načina, kroz korisničko sučelje ili koristeći *T-SQL* programski jezik. Radi preglednosti i bolje čitljivosti, u radu će biti prikazano kako se dostupne grupe kreiraju koristeći korisničko sučelje.

Unutar alata za upravljanje *SQL* bazama podataka pod nazivom *SQL Server Management Studio*, najprije se treba povezati na SQL instancu koja će predstavljati primanu repliku. Za potrebe ovoga rada kreirane su dvije baze podataka pod nazivom ZavrsniRad_VG i ZavrsniRad_VG2, čije će strukture biti opisane u poglavlju 6.1 (Baza podataka - *SQL Server*). Kako bi se kreirala nova dostupnu grupa, potrebno je pozicionirati se na *Availability Groups* podmapu unutar mape *AlwasyOn High Availability* te tamo odabrati kreiranje nove grupe kroz *New Availability Group Wizard*, kako je prikazano na slici (Slika 4.5).



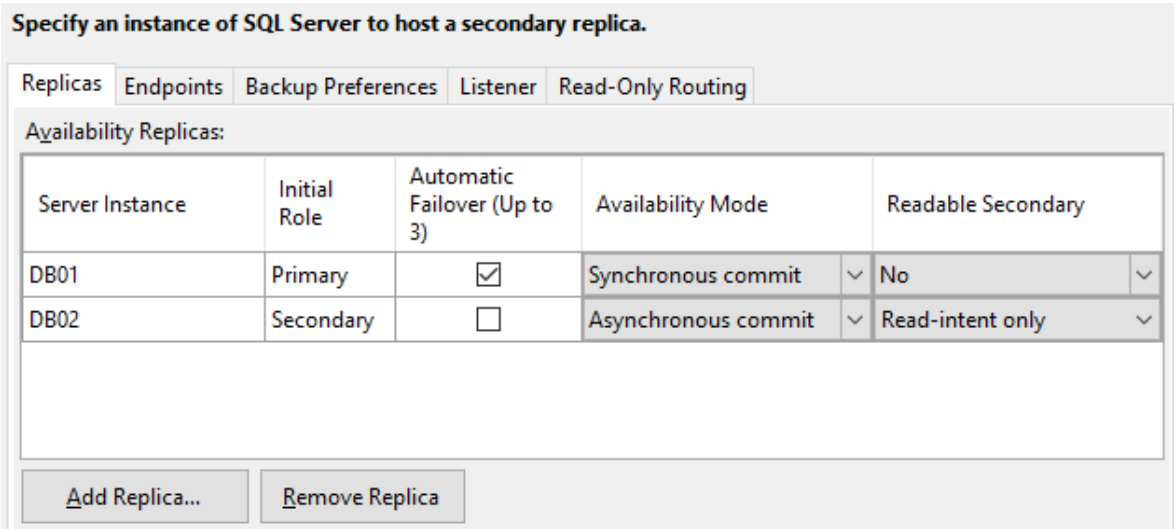
Slika 4.5 Kreiranje nove dostupne grupe

Nakon toga se odabire ime dostupne grupe koje će se koristiti te vrsta klastera (Slika 4.6).



Slika 4.6 Imenovanje dostupne grupe i ostale postavke

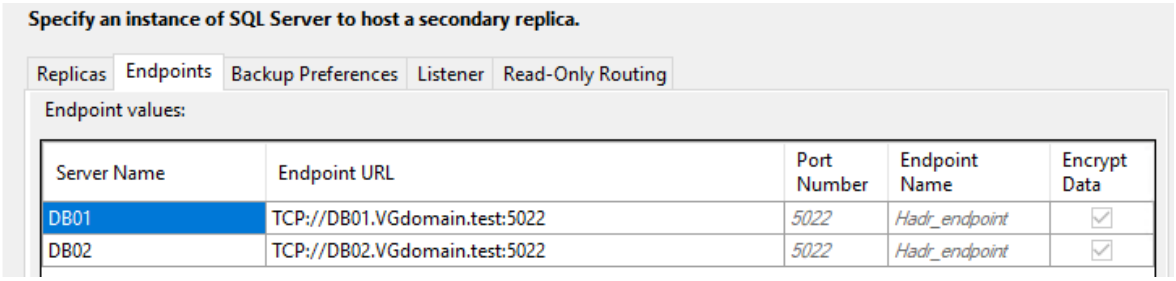
Detekcija ispravnosti na razini baze (engl. *Database Level Health Detection*) je postavka koja određuje hoće li se dostupna grupa prebaciti na sekundarni poslužitelj u slučaju nedostupnosti bilo koje baze koja sudjeluje u dostupnoj grupi.[42] Kako bi se dovršilo kreiranje nove dostupne grupe, čarobnjak za instalaciju provjerava jesu li ispunjeni svi preduvjeti. Tako mora postojati barem jedna sigurnosna kopija (engl. *full backup*) te sve baze koje sudjeluju u grupi moraju biti u *full recovery* načinu rada. Nakon toga još preostaje podesiti načine rada svake pojedine replike (Slika 4.7).



Slika 4.7 Odabir načina rada replika

[42] https://www.sqlservercentral.com/blogs/configure-always-on-ag-with-sql-2017; 21.11.2019.

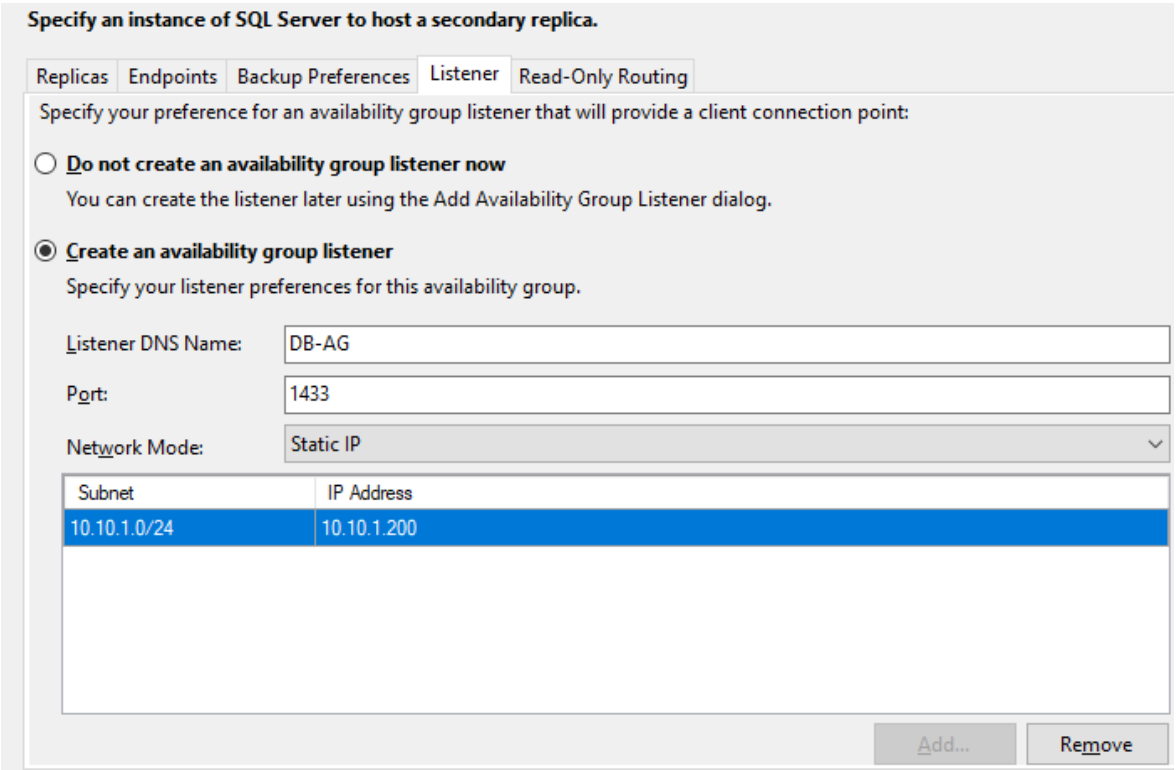
Za svaku instancu koja sudjeluje u dostupnoj grupi postoji mogućnost izbora radi li se o primarnoj ili sekundarnoj instanci, radi li s automatskim prebacivanjem baza te radi li u sinkronom ili asinkronom načinu rada. U podizborniku *Endpoints* podešavaju se pristupne točke preko kojih se međusobno povezuju *SQL* instance koje sudjeluju u dostupnim grupama (Slika 4.8).



Slika 4.8 Podešavanje pristupnih točaka replika

SQL nudi predefinirane vrijednosti portova (5022) te naziv pristupne točke (engl. *endpoints*), koje je po želji moguće izmijeniti. Predefinirano je uključena enkripcija podataka koji prometuju između replika, što je moguće isključiti.

Kako bi se završilo kreiranje dostupne grupe, preostalo je još kreirati *AG listener* koji predstavlja klijentsku pristupnu točku dostupne grupe. Ta pristupna točka ima definirano ime, IP adresu na razini svake podmreže kojoj pripada te port na kojem sluša (Slika 4.9).



Slika 4.9 Definiranje *AG listenera*

Aplikacije za pristup dostupnoj grupi upravo i koriste taj naziv i *port* umjesto pravih naziva i *portova* poslužitelja, jer se aplikacija spaja na virtualno ime *AG listenera*.

Preostaje još podesiti usmjeravanje na replike za čitanje (engl. *read-only routing*), o čemu će biti posvećeno poglavlje 5.2 (Konfiguriranje sekundarne replike za čitanje). Nakon što je sve podešeno, može se pokrenuti inicijalna sinkronizacija podataka s primarne na sekundarnu repliku, koristeći jedan od sljedećih načina:

- ***Automatic Seeding način*** podrazumijeva automatsko kreiranje baza za svaku sekundarnu repliku uz pretpostavku da su iste putanje podatkovnih i zapisničkih datoteka na svim instancama.
- ***Full database and log backup način*** radi tako da započinje sinkronizaciju izvođenjem potpunog *backupa* i *backupa* zapisa za svaku odabranu bazu. Te se baze zatim uspostavljaju na sekundarnim replikama i dodaju se u dostupnu grupu.
- ***Join only način*** pretpostavlja da već postoje uspostavljene kopije baze i zapisa na sekundarnom poslužitelju te se samo dodaju u dostupnu grupu.

Na kraju ovog postupka, u potpunosti je spremna dostupna grupu nazvana *AG-ZavrsniRad*, kojoj se pristupa koristeći *AG listener* naziv DB-AG.

4.4. Administracija *AlwaysOn* sustava

Upravljanje klasterom

Iako je klaster inicijalno konfiguriran i dalje postoji potreba za periodičkim održavanjem klastera. Tako npr. postoji potreba za prebacivanjem instanci *SQL Servera* na drugi čvor u klasteru. Za ovim često postoji potreba prilikom ažuriranja zakrpi operativnog sustava ili *SQL Servera*. Preporuka je da se najprije ažuriraju zakrpe na pasivnom čvoru klastera pa tek ako sve prođe bez problema i na aktivnom čvoru. Tek onda je moguće sa sigurnošću obaviti prebacivanje baza na poslužitelj koji je inicijalno bio primarni. Radnje prebacivanja instance *SQL Servera* na drugi čvor moguće je izvršiti unutar *Failover Cluster Managera*.

Upravljanje dostupnim grupama

Nakon inicijalne instalacije dostupne grupe se moraju održavati. Tako je ponekad potrebno prebaciti dostupne grupe na sekundarni poslužitelj te ih redovno nadzirati. Prebacivanje dostupnih grupa na sekundarnu repliku u sinkronom načinu rada može se obaviti izvršenjem *T-SQL* naredbe (Programski isječak 4.1).

```
ALTER AVAILABILITY GROUP [AG-ZavrsniRad] FAILOVER;
```

Programski isječak 4.1 Naredba za prebacivanje dostupnih grupa u sinkronom načinu rada

Ako se radi o dostupnoj grupi koja radi u asinkronom načinu rada, postupak je nešto drugačiji; moguće je pokrenuti naredbu za prebacivanje na sekundarnu repliku uz mogućnost gubitka podataka (Programski isječak 4.2).

```
ALTER AVAILABILITY GROUP [AG-ZavrsniRad]
FORCE_FAILOVER_ALLOW_DATA_LOSS;
```

Programski isječak 4.2 Naredba za prebacivanje dostupnih grupa u asinkronom načinu rada

Ostale administrativne radnje

U nekim slučajevima baze podatka se moraju staviti u način rada koji omogućuje spajanje samo jednog korisnika (engl. *single-user mode*), ili u način za čitanje (engl. *read-only mode*), što nije moguće u slučaju kada baza sudjeluje u dostupnoj grupi. Iz tog razloga ponekad je potrebno privremeno pauzirati sinkronizaciju baza dostupne grupe dok ne obavimo potrebne radnje ili čak u potpunosti bazu izbaciti iz dostupne grupe. Sinkronizacija baza u dostupnoj grupi se zaustavlja, privremeno pauzira, odnosno nastavlja pomoću naredbi, kako je prikazano u programskom isječku (Programski isječak 4.3).

```
ALTER DATABASE [ZavrsniRad_VG] SET HADR OFF ;
GO
ALTER DATABASE [ZavrsniRad_VG] SET HADR SUSPEND;
GO
ALTER DATABASE [ZavrsniRad_VG] SET HADR RESUME;
GO
```

Programski isječak 4.3 Zaustavljanje, pauziranje i nastavak sinkronizacije podataka baze u dostupnoj grupi

5. Sekundarna replika za čitanje

5.1. Prednosti i nedostaci sekundarne replike

Sekundarna replika u *AlwaysOn* sustavu, osim što sudjeluje u ostvarenju redundantnosti tako što predstavlja dodatnu kopiju produkcijskih podataka, može poslužiti i kao aktivna kopija za rasterećenje produkcijskog opterećenja (engl. *production workload*). Tako sekundarne replike mogu poslužiti kao sustav izvještavanja, ako *read-only* upite aplikacije preusmjerimo na repliku za čitanje. Osim toga, sekundarna replika može poslužiti i za izvođenje radnji održavanja baza podataka, kao što su izrade sigurnosne kopije te provjere integriteta baza podataka. Time se znatno rasterećuju produkcijski poslužitelji te značajno povećavaju performanse poslovno kritičnih aplikacija (engl. *mission-critical applications*), reducirajući opterećenje procesora i diskovnog sustava uzrokovanog izradom sigurnosnih kopija.[43]

Kao nedostatak sekundarne replike, za čitanje se može navesti jedino dodatni trošak za licence *SQL Servera* jer, sve dok ne koristimo sekundarnu repliku za bilo kakve aktivnosti (iako samo za izradu *backupa* ili kao *reporting* sustav), ne trebamo dodatnu licencu.

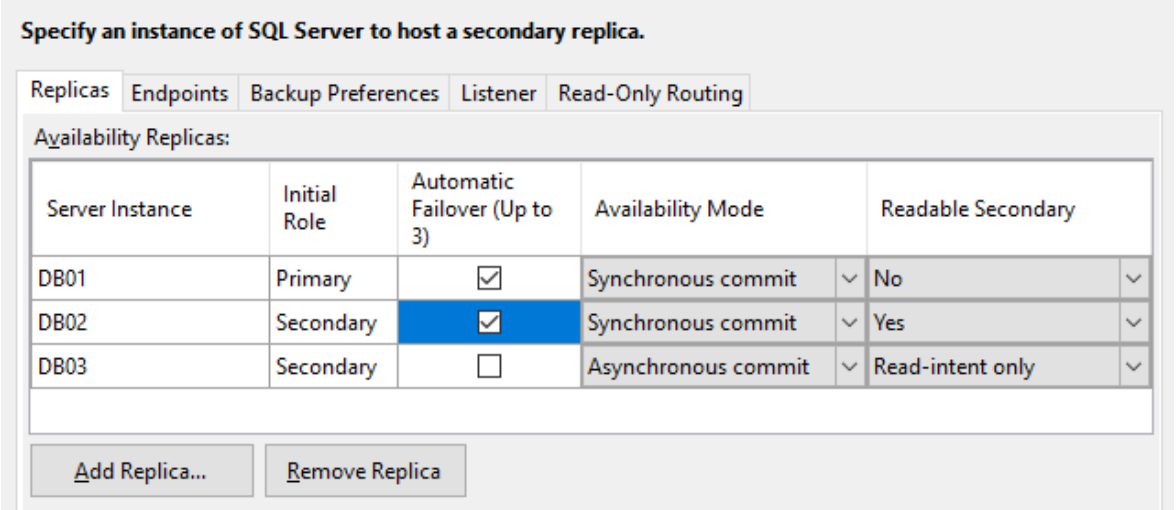
5.2. Konfiguriranje sekundarne replike za čitanje

Konfiguriranje sekundarne replike za čitanje moguće je izvesti na više načina: korištenjem korisničkog sučelja unutar *SQL Server Management Studia*, pisanjem *T-SQL* naredbi ili koristeći *Powershell*.

Za dodavanje nove *read-only* replike (DB03), najprije je potrebno na razini instance omogućiti *AlwaysOn*, kao što je to učinjeno i za prethodno konfigurirane dvije instance (DB01 i DB02). Ovaj postupak je već opisan u poglavlju 4.3 (Uspostavljanje i konfiguracija). Nakon što je na razini instance omogućen *AlwaysOn*, može se pristupiti dodavanju nove sekundarne replike u postojeću dostupnu grupu, tako da se unutar *SQL Server Management Studia* poveže na primarnu repliku (DB1), unutar foldera *Availability Groups* i označi postojeću dostupnu grupu (AG-ZavrsniRad) te desnim klikom odabere *Add Replica*. Tada se otvara prozor za konfiguraciju gdje se upisuje naziv sekundarne replike te

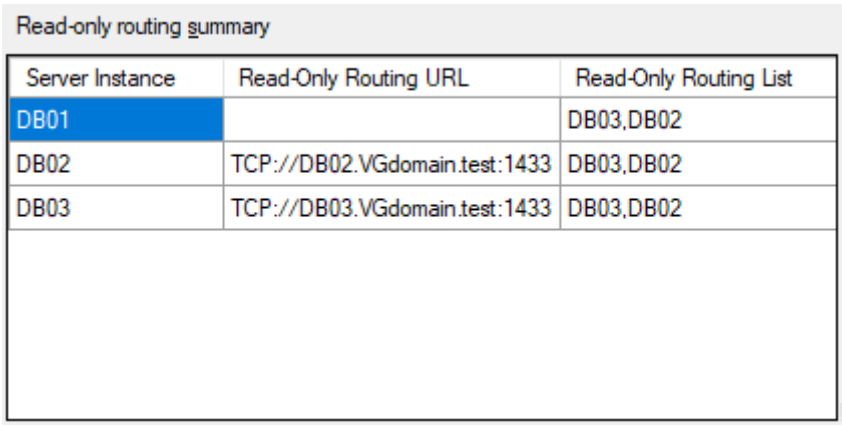
[43] Pro SQL Server Always On Availability Groups; Uttam Parui, Vivek Sanil; str. 163.

odabire hoće li raditi u sinkronom ili asinkronom načinu rada, hoće li raditi s automatskim prebacivanjem dostupnih grupa te radi li se o replici za čitanje ili ne, kako je prikazano na slici (Slika 5.1).



Slika 5.1 Konfiguriranje sekundarne replike za čitanje

Aplikacija se može spajati na repliku za čitanje koristeći pravi naziv instance ili koristeći *AG Listener* naziv. Kako bi aplikacija mogla koristiti mogućnost automatskog preusmjeravanja na replike za čitanje, koju omogućuju *AlwaysOn* dostupne grupe, aplikacija mora koristiti *AG Listener* naziv. Stoga je još potrebno konfigurirati *URL* za repliku za čitanje (engl. *read-only URL*) te listu za preusmjeravanje upita za čitanje (engl. *read-only routing list*), kako je prikazano na slici (Slika 5.2).



Slika 5.2 Dodavanje *read-only routing* liste

Time se daje informaciju *AlwaysOn* sustavu na koju repliku da proslijedi upit, ako je unutar aplikacije konekcija konfigurirana da ima *read-only intent*. O tome će biti više riječi u poglavlju 6.4 (Usmjeravanje aplikacije na repliku za čitanje). Konekciji koja je konfigurirana tako da ima postavljen *read-only intent*, nije dopušteno povezivanje na primarnu repliku.

Time se sprječava korisnike da se greškom spoje na primarnu repliku kada iniciraju read-only zahtjev.[44] Postavke i načine rada replika moguće je i naknadno izmijeniti izvršavanjem *T-SQL* naredbi na primarnoj replici, kao što je prikazano u programskom isječku (Programski isječak 5.1).

```
ALTER AVAILABILITY GROUP [AG-ZavrsniRad]
MODIFY REPLICA ON N'DB03' WITH
(SECONDARY_ROLE(
ALLOW_CONNECTIONS = READ_ONLY,
READ_ONLY_ROUTING_URL = N'TCP://DB03.VGdomain.test:1433'));
```

Programski isječak 5.1 Naredbe za prebacivanje replike u *read-only* način te konfiguriranje *read-only routinga*

5.3. Distribucija opterećenja između više replika za čitanje

Ako postoji više replika za čitanje, *read-only routing* lista se konfigurira tako da se upišu replike za čitanje odvojene zarezom, kao što je prikazano u programskom isječku, uz pretpostavku da su navedene replike konfigurirane za čitanje. Mogu se čak formirati i grupe lista odvojene u zasebne zagrade, pa se u tom slučaju najprije izvršava preusmjeravanje na replike iz prve liste, a tek ako one nisu dostupne na replike iz druge liste. (Programski isječak 5.2).

```
USE MASTER
GO
ALTER AVAILABILITY GROUP [AG-ZavrsniRad]
MODIFY REPLICA ON N'DB01' WITH
(PRIMARY_ROLE (READ_ONLY_ROUTING_LIST=(('DB02','DB03'),
('DB04','DB05'))));
```

Programski isječak 5.2 Konfiguriranje distribucije opterećenja na *read-only routing* listi

Distribucija opterećenja uvedena je tek sa *SQL Serverom 2016*, a pri tom koristi *raund-robin metodu* distribucije *read-only* konekcija. Tako da se svaka sljedeća konekcija preusmjerava na sljedeću grupu replika za čitanje, odnosno na sljedeću repliku iz liste.

[44]https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/configure-read-only-access-on-an-availability-replica-sql-server?view=sql-server-2017; 16.12.2019.

6. Izrada web aplikacije za demonstraciju rada s *AlwaysOn* sustavom

Za potrebe demonstracije rada s bazama podataka kojima je osigurana visoka dostupnost primjenom *AlwaysOn* tehnologije, kreirana je *web* aplikacija koristeći *ASP.NET MVC* (engl. *Model-View-Controller*) tehnologiju. Aplikacija predstavlja blagajnu za izdavanje računa te sadrži osnovne funkcionalnosti, poput registracije i prijave, a izvršava operacije upisa, ispisa, ažuriranja i brisanja (engl. *create, read, update, delete*, skraćeno *CRUD*) nad podacima u bazi te sadrži izvještaje koji koriste *read-only* replike. Aplikacija je kreirana koristeći razvojni alat *Microsoft Visual Studio 2017*, a za razvoj baze podataka korišten je *SQL Server Management Studio 2017*.

6.1. Baza podataka - *SQL Server*

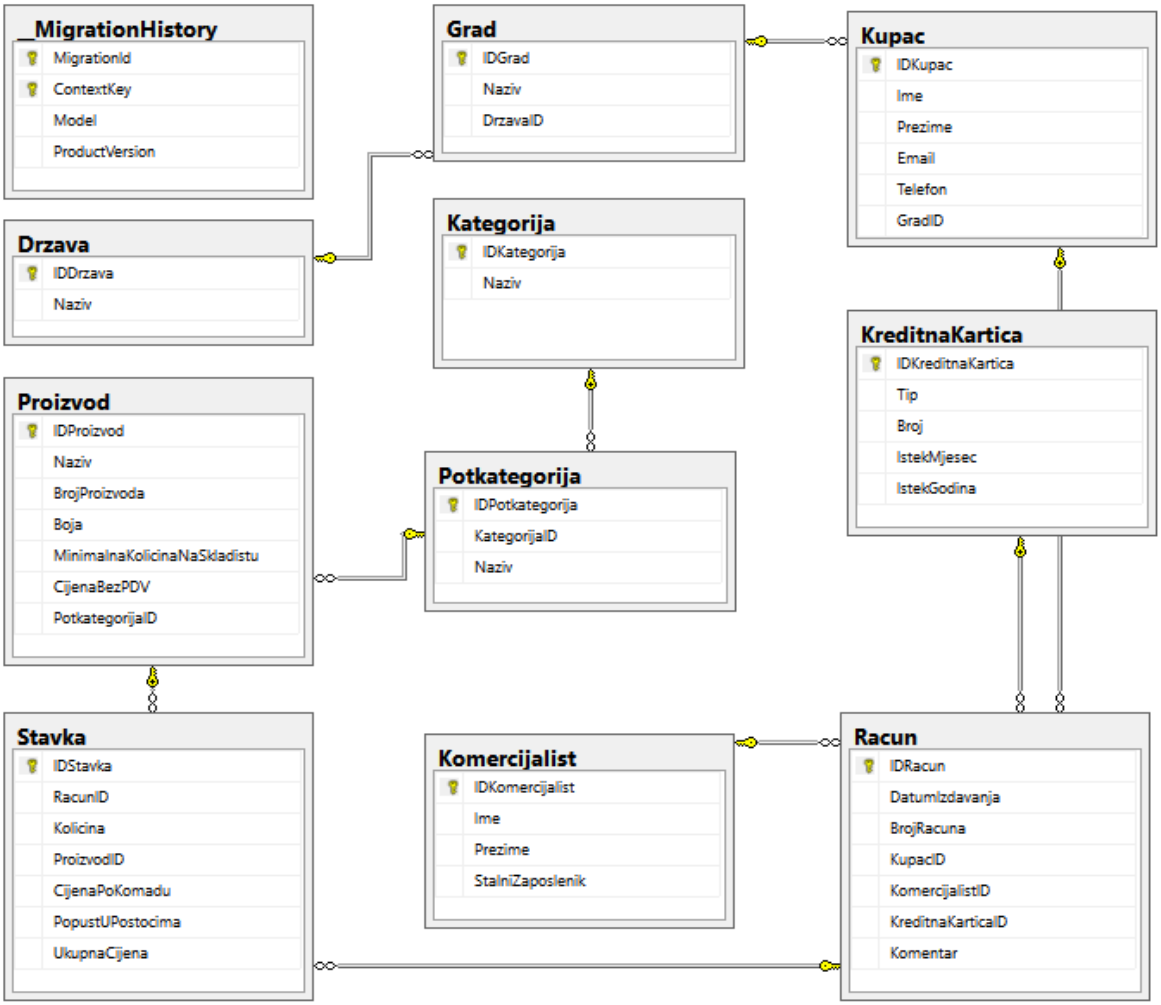
Kao sustav za upravljanje bazama podataka (engl. *Relational database management system*, skraćeno RDBMS) korišten je *Microsoft SQL Server 2017* koji je sa sobom donio znatna poboljšanja i ubrzanja *AlwaysOn* sustava u odnosu na prethodne verzije. Za potrebe *web* aplikacije kreirane su dvije baze pod nazivom ZavrsniRad_VG i ZavrsniRad_VG2 kako bi se simulirala njihova poslovna povezanost te tako stvorila potreba da se nalaze u istoj dostupnoj grupi. Te dvije baze su međusobno povezane tako da se u bazi ZavrsniRad_VG nalazi uskladištena procedura `[dbo].[RptRacuniSumarno]` koja čita podatke iz jedne i druge baze te vraća rezultat zajedničkog sumiranja i grupiranja podataka.

Baza ZavrsniRad_VG sastoji se od ukupno 16 entiteta koji su podijeljeni u 2 grupe

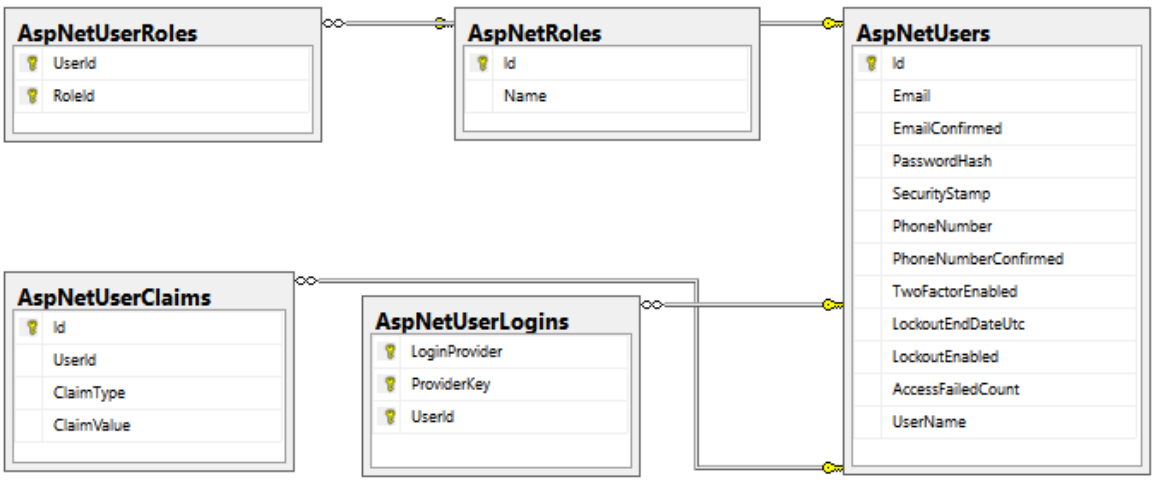
- grupa entiteta poslovnih podataka aplikacije
- grupa entiteta iz *ASP.NET - Individual Authentication* predloška (engl. *template*)

U poslovne entitete (Kupac, Racun, Stavka...) aplikacija zapisuje podatke te ih iz istih i dohvaća te prikazuje. Drugim riječima, obavlja s njima osnovne *CRUD* operacije (Slika 6.1).

Entiteti koji su kreirani iz *ASP.NET - Individual Authentication* predloška sadrže podatke o korisnicima aplikacije koji služe za autentikaciju i autorizaciju korisnika (Slika 6.2).



Slika 6.1 Dijagram baze podataka ZavrsniRad_VG - poslovni entiteti[45]



Slika 6.2 Dijagram baze podataka ZavrsniRad_VG - entiteti za autentikaciju i autorizaciju[46]

[45] Izvor: Vlastiti rad autora; 3.1.2020.
[46] Ibidem; 3.1.2020.

Kako bi se unutar aplikacije moglo prikazati na kojem poslužitelju je upit izvršen, s obzirom na to da aplikacija koristi isključivo *AG Listener* naziv za povezivanje na bazu, u bazi *ZavrsniRad_VG* kreirana je uskladištena procedura pod nazivom `[dbo].[DohvatiNazivServera]` koja kao rezultat prikazuje naziv instance na kojoj je izvršena, kao što je prikazano u programskom isječku (Programski isječak 6.1).

```
USE [ZavrsniRad_VG]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[DohvatiNazivServera]
AS
SELECT @@Servername as ServerName
GO
```

Programski isječak 6.1 Uskladištena procedura koja vraća naziv poslužitelja na kojem je izvršena

Kao dodatak radu bit će priložene sve skripte korištene za potrebe ovoga rada za kreiranje baza podataka i njihovih objekata.

6.2. Web aplikacija - *ASP.NET MVC* tehnologija

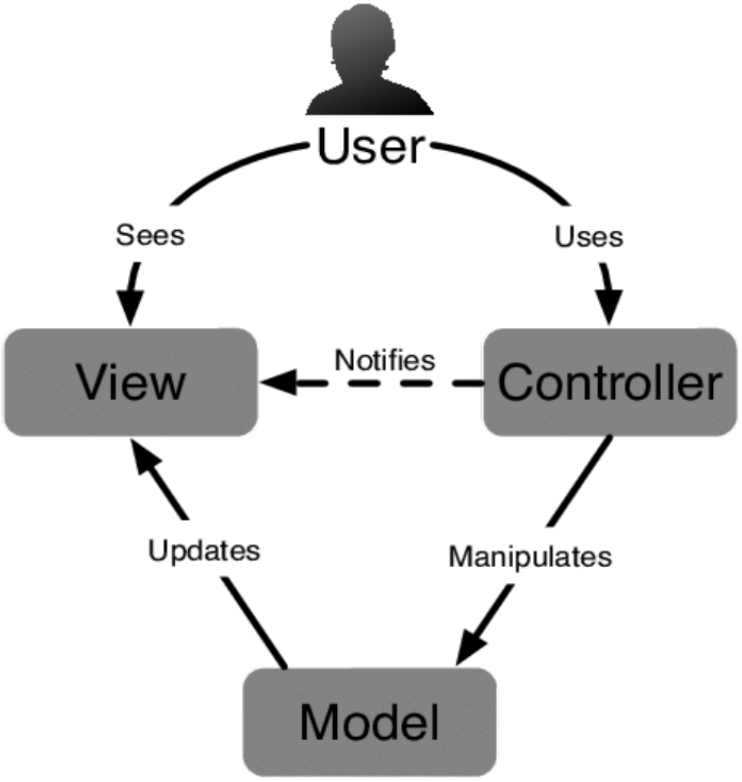
Za izradu *web* aplikacije koja u osnovi predstavlja blagajnu za izradu računa, korišten je *ASP.NET MVC* okvir (engl. *Framework*) te je primijenjen *Model-View-Controller* obrazac (engl. *pattern*) koji odvaja programsku logiku u tri povezana elementa:[47]

- **Model (**engl. *Model***)** predstavlja dio aplikacije koji implementira podatkovnu logiku te dohvaća i sprema stanje modela u bazu podataka. Prima korisničke zahtjeve od upravitelja.
- **Pogled (**engl. *View*) je komponenta aplikacije koja prikazuje korisničko sučelje aplikacije. Služi za prikaz podataka modela prema korisniku u određenom formatu (npr. graf, dijagram, tablica...).

[47] https://www.w3resource.com/asp.net/mvc-architecture.php; 15.1.2020.

- **Upravitelj (**engl. *Controller*) obrađuje i odgovara na korisničke zahtjeve. Radi s modelom te određuju koji će pogled biti prikazan.

Ove tri komponente sudjeluju u međusobnoj interakciji, kako je prikazano na slici (Slika 6.3).



Slika 6.3 Prikaz međusobne interakcije komponenti unutar MVC uzorka[48]

Kako bi se ovaj obrazac primijenio na aplikaciju, prilikom izrade *web* aplikacije odabran je unutar razvojnog alata *Visual Studio 2017* predložak *ASP.NET Web Application* te je unutar njega odabrana MVC arhitektura. Dodatno je za vrstu autentikacije uključena opcija *Individual User Account,* kako bi se unutar aplikacije implementirao *Microsoft. NET Identity* okvir koji je zadužen za autentikaciju i autorizaciju unutar aplikacije.

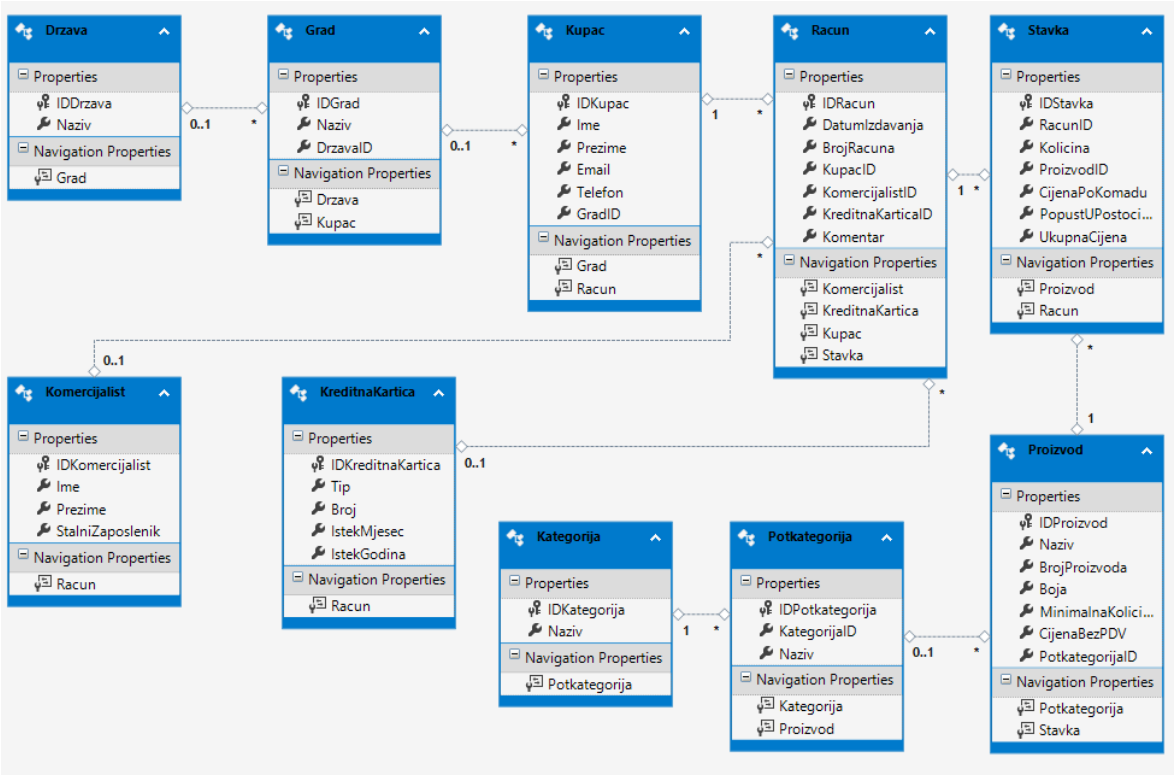
Kako bi se izmijenilo pretpostavljeno korištenje *LocalDb* baze za *Microsoft .NET Identity* okvir, dodatno je u *Web.config* datoteci izmijenjen *connection string*, tako da koristi *SQL Server* kao *RDMBS*. Stoga je u *connection string* upisan *AG Listener* naziv (DB-AG).

[48] https://www.researchgate.net/figure/Model-view-controller-architecture_fig7_320249584; 19.1.2020.

Modeli

Objekti modela unutar aplikacije primaju i pohranjuju stanje modela u bazu podataka.[49] Model omogućava i spremanje vrijednosti koje se koriste u pogledu, tako da se u model mogu spremiti entiteti koji su građeni od atributa, a koji apstraktno opisuju objekt. S obzirom na to da je korišten *Entity Framework*, a ne klasični model, kreiran je *Entity Data Model* za postojeću bazu koristeći *database-first* pristup. Radi se o modelu koji opisuje entitete i relacije između njih. Kreiran je tako da je unutar projekta u *Visual Studiu* odabrano dodavanje nove stavke te je potom odabran *ADO.NET Entity Data Model*. U nastavku je izabrana opcija *EF Designer from database* te je korišten *Entity Framework 6.0*. Nakon toga su u model dodani svi potrebni entiteti te uskladištene procedure iz baze ZvrsniRad_VG.

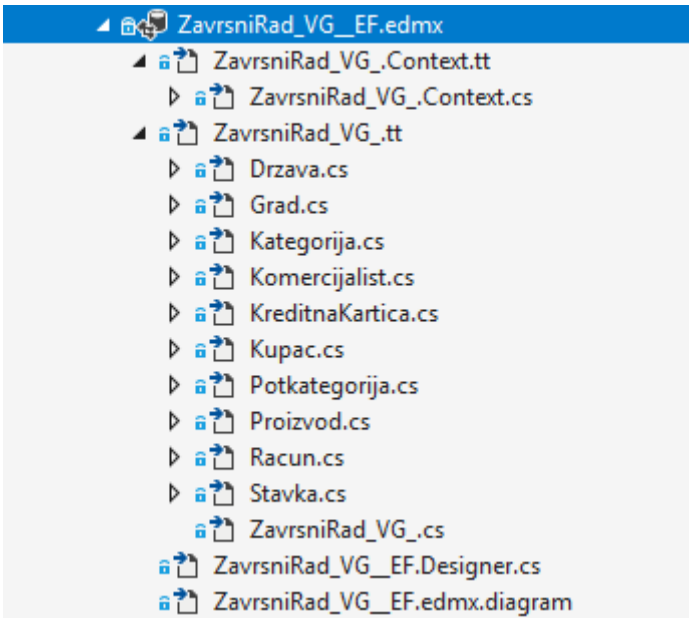
Entity Framework je generirao model iz entiteta baze ZavrsniRad_VG te je kreirao sve relacije između entiteta (Slika 6.4).



Slika 6.4 Dijagram modela sa svojstvima i relacijama između entiteta

Ujedno su unutar direktorija *Model* automatski kreirane datoteke povezane s modelima generiranim iz baze. Kreirani su i *context* i *entity* razredi (engl. *class*) za svaki entitet iz baze, kako je prikazano na slici (Slika 6.5).

[49] https://www.w3resource.com/asp.net/mvc-architecture.php; 15.1.2020.



Slika 6.5 Model kreiran iz baze

ZavrsniRad_VG.Context.cs datoteka sadrži razred *ZavrsniRad_VG_Entities* koji nasljeđuje *DbContext* razred *Entity Frameworka* te sva njegova svojstva i metode. U njemu su definirana svojstva za svaki razred modela, kako je prikazano u programskom isječku (Programski isječak 6.2).

```
public partial class ZavrsniRad_VG_Entities : DbContext
{
      public ZavrsniRad_VG_Entities()
            : base("name=ZavrsniRad_VG_Entities")
}
public DbSet<Drzava> Drzava { get; set; }
public DbSet<Grad> Grad { get; set; }
public DbSet<Kategorija> Kategorija { get; set; }
public DbSet<Komercijalist> Komercijalist { get; set; }
public DbSet<KreditnaKartica> KreditnaKartica { get; set; }
public DbSet<Kupac> Kupac { get; set; }
public DbSet<Potkategorija> Potkategorija { get; set; }
public DbSet<Proizvod> Proizvod { get; set; }
public DbSet<Racun> Racun { get; set; }
public DbSet<Stavka> Stavka { get; set; }
```

Programski isječak 6.2 Programski isječak strukturu modela

Za svaki od entiteta postoji zaseban razred u kojem su definirana svojstva, kako je prikazano u programskom isječku (Programski isječak 6.3 Programski isječak razred).

```
public partial class Kupac
{
    public Kupac()
    {
        this.Racun = new HashSet<Racun>();
    }
    public int IDKupac { get; set; }
    public string Ime { get; set; }
    public string Prezime { get; set; }
    public string Email { get; set; }
    public string Telefon { get; set; }
    public Nullable<int> GradID { get; set; }
    public virtual Grad Grad { get; set; }
    public virtual ICollection<Racun> Racun { get; set; }
}
```

Programski isječak 6.3 Programski isječak razreda Kupac

Pogledi

Pogledi su kreirani koristeći *Razor (CSHTML)* sintaksu koja se temelji na *C#* programskom jeziku. Radi se o načinu stvaranja dokumenta programskim kodom s klijentske strane, kako je prikazano u programskom isječku iz pogleda *Reports* (Programski isječak 6.4).

```
@{
    ViewBag.Title = "Index";
}
<h2>Računi po godinama</h2>
<h2 id="ServerName"></h2>
<table class="table table-striped table-bordered table-hover
table-responsive-sm table-responsive-md table-responsive-lg
table-responsive-xl">
    <tr>
        <th>
            Godina
        </th>
        <th>
            Iznos
        </th>
    </tr>
    @foreach (var item in Model)
    {
        <tr>
            <td>
                @item.Godina
            </td>
            <td>
                @item.Iznos
            </td>
        </tr>
    }
 </table>
```

Programski isječak 6.4 Primjer korištenja razor sintakse unutar pogleda

Kako bi se stvorio dio sadržaja koji je identičan na svim stranicama aplikacije, najprije je kreiran globalni predložak (engl. *Layout*) koji dodaje zaglavlje i podnožje na svim ostalim pogledima. Budući da pogledi podržavaju i ostale klijentske programske jezike i biblioteke, dodatno su iskorištene mogućnosti koje pružaju *JavaScript* i *AJAX*. S obzirom na to da aplikacija koristi *AG Listener* naziv za povezivanje na *SQL Server* pa se ne može znati na koju instancu *SQL Servera* će biti preusmjerena, osmišljen je mehanizam koji će prikazivati stvarni naziv instance *SQL Servera*. U tu svrhu iskorišten je *AJAX* poziv unutar *JavaScript* skripte, kako je prikazano u programskom isječku (Programski isječak 6.5).

```
<script type="text/javascript">
    function RefreshStats() {
        $.ajax({
            cache: false,
            type: "GET",
            url: "/Statistike/ServerName",
            success: function (result) {
                  $("#ServerName").html("<p>Naziv servera: "
                  + result.NazivServera + "</p>");
            }
        });
       }
      setInterval(RefreshStats, 1000);
</script>
```

Programski isječak 6.5 Programski isječak s prikazom *AJAX* poziva

AJAX poziv izvršava se svake sekunde te dohvaća naziv *SQL* instance. Dodatno mu je postavljeno svojstvo `cache` na vrijednost `false` kako se naziv instance ne bi dohvaćao iz predmemorije, već svaki put iz baze.

Upravitelji

Upravitelji u *MVC* arhitekturi upravljaju sa svim dolaznim *URL* zahtjevima, a predstavljaju razred proistekao iz osnovnog razreda (engl. *base class*) *System.Web.Mvc.Controller*.[50] Zaprimaju zahtjeve, dohvaćaju podatke iz baze te ih spremaju u model, a nakon izvršenih operacija nad njima, vraćaju odgovor. Stoga kažemo da upravitelji predstavljaju vezu između modela i pogleda. Upravitelji izvršavaju različite akcije pomoću svojih akcijskih metoda (Programski isječak 6.6).

[50] https://www.tutorialsteacher.com/mvc/mvc-controller; 20.12.2019.

```
public ActionResult Details(int? id)
{
	if (id == null)
	{
		return new
		HttpStatusCodeResult(HttpStatusCode.BadRequest);
	}
	Kupac kupac = db.Kupac.Find(id);
	if (kupac == null)
	{
		return HttpNotFound();
	}
	return View(kupac);
}
```

Programski isječak 6.6 Akcijske metode upravitelja

Svi se upravitelji moraju nalaziti u direktoriju *Controllers* MVC strukture te svi završavaju sufiksom *Controller* (npr. *RacunController*, *StavkaController*)[51].

6.3. *CRUD* operacije - *Entity Framework 6* i *ASP.NET MVC 5* tehnologija

Kako bi aplikacija komunicirala s bazom podataka te izvršavala *CRUD* operacije, korišteno je objektno-relacijsko mapiranje objekata (engl. *object-relational mapping*, skraćeno ORM). Implementiran je Microsoftov ORM okvir (engl. *framework*) *Entity Framework* verzije 6, primjenjujući pristup prvotnog kreiranja baze podataka (engl. *database-first approach*). Kako bi se unutar aplikacije omogućilo izvršavanje osnovnih *CRUD* operacija, prilikom kreiranja upravitelja odabran je kao vrsta upravitelja *MVC 5 Controller with views, using Entity Framework.* Tako su kreirani upravitelji i pogledi koji već imaju implementirane funkcionalnosti za unos, ažuriranje, kreiranje i prikazivanje podataka modela. Kako bi upravitelj znao komunicirati s bazom, na početku mu je deklariran i instanciran *Entity Framework model* te ga u nastavku može koristiti za *CRUD* operacije, kao što je navedeno u programskom isječku iz upravitelja *KupacController* na primjeru operacije unosa novog kupca u bazu (Programski isječak 6.7).

[51] Ibidem; 20.12.2019.

```
private ZavrsniRad_VG_Entities db = new
ZavrsniRad_VG_Entities();

public ActionResult Create()
        {
            ViewBag.GradID = new SelectList(db.Grad,
            "IDGrad", "Naziv");
            return View();
        }
[HttpPost]
[ValidateAntiForgeryToken]
[Authorize(Roles = "Admin")]

 public ActionResult Create([Bind(Include =
"IDKupac,Ime,Prezime,Email,Telefon,GradID")] Kupac kupac)
{
   if (ModelState.IsValid)
      {
        db.Kupac.Add(kupac);
        db.SaveChanges();
        return RedirectToAction("Index");
      }
      ViewBag.GradID = new SelectList(db.Grad,
      "IDGrad","Naziv", kupac.GradID);

   return View(kupac);
}
```

Programski isječak 6.7 Primjer akcijske funkcije za unos podataka u bazu

Ovaj kod dodaje entitet Kupac, koji je kreirao *ASP.NET MVC model binder*, u *entitiy set* Kupac i sprema promjene u bazu.[52]

6.4. Usmjeravanje aplikacije na repliku za čitanje

S obzirom na to da je preusmjeravanje prometa na replike za čitanje zadatak *AlwaysOn* sustava i *Windows Failover Clustera*, unutar aplikacije su potrebne minimalne izmjene.

[52]https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/implementing-basic-crud-functionality-with-the-entity-framework-in-asp-net-mvc-application; 20.12.2019.

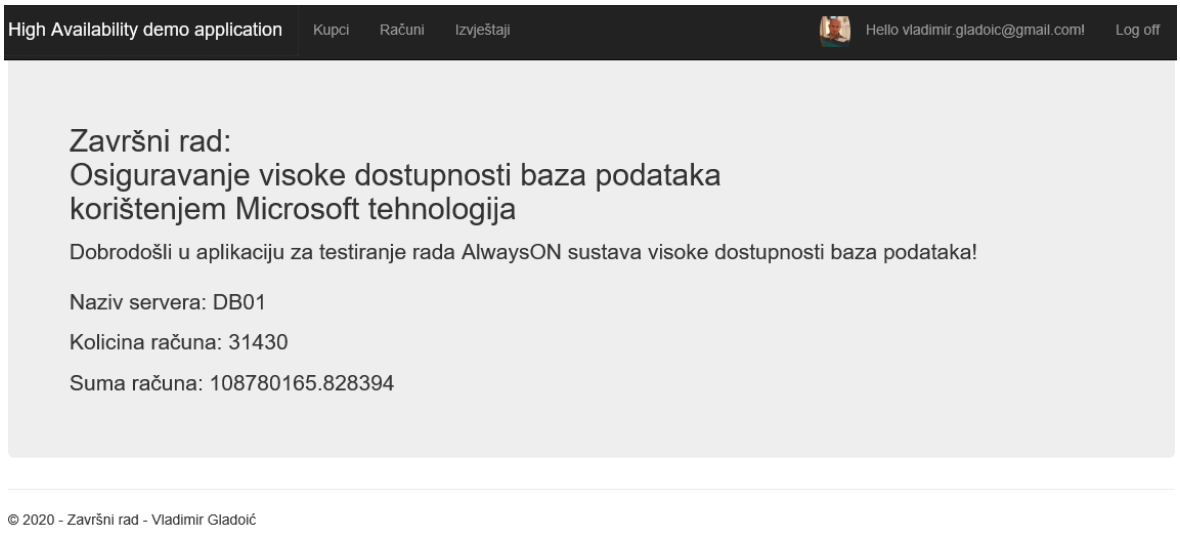
Potrebno je najprije unutar *Web.config* datoteke zamijeniti naziv poslužitelja s *AG Listener* nazivom. Tako *AlwaysOn* sustav može sam odlučiti na koju repliku će proslijediti upit aplikacije. Ako se pojedini dijelovi aplikacije žele preusmjeriti isključivo na repliku za čitanje, potrebno je u *connection string* dodati obilježje `ApplicationIntent=ReadOnly`. Aplikacija tako uopće nije svjesna na koji poslužitelj odlazi njen upit.

6.5. Funkcionalnosti aplikacije

Web aplikacija koja je izrađena za potrebe demonstracije rada s *AlwaysOn* dostupnim grupama, u osnovi predstavlja blagajnu za izradu računa. Sadrži i osnovne funkcionalnosti za pregled, dodavanje i uređivanje podataka o kupcu te računima. U svrhu demonstracije rada s replikama za čitanje, kreiran je izbornik pod nazivom "Izvještaji", čiji upiti su preusmjereni na replike za čitanje. Kako bi aplikaciju mogli koristiti samo prijavljeni korisnici te kako bi pojedine radnje mogli izvršiti samo korisnici s odgovarajućim pravima, implementiran je i obrazac za prijavu i registraciju korisnika.

Početna stranica

Na početnoj stranici, kojoj mogu pristupiti prijavljeni ili neprijavljeni korisnici, nalaze se osnovne informacije o aplikaciji, kao što su naziv poslužitelja na kojem se upiti izvršavaju te količina i suma računa na poslužitelju. U zaglavlju aplikacije, koje je zajedničko svim stranicama aplikacije, nalaze se izbornici aplikacije te poveznica (engl. *link*) za prijavu i odjavu iz aplikacije (Slika 6.6).

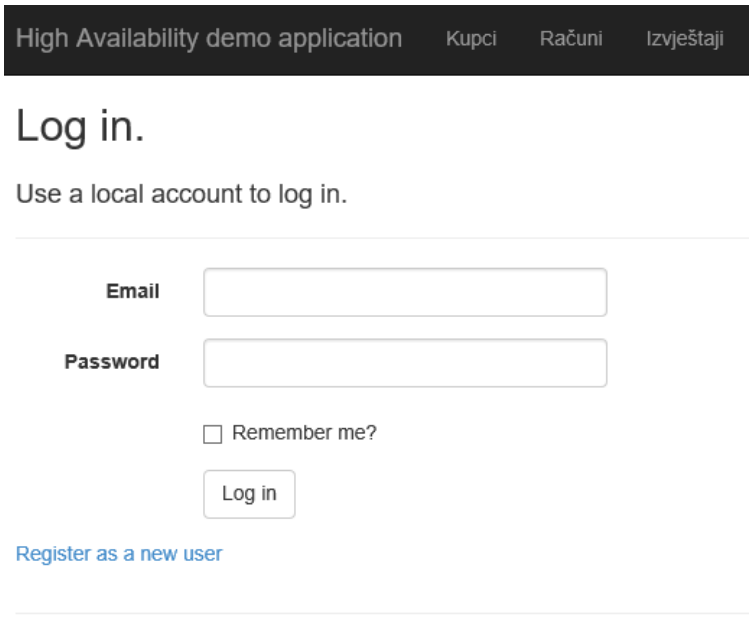


Slika 6.6 Početna stranica aplikacije

Na početnoj stranici se svaku sekundu osvježavaju podaci o nazivu poslužitelja, količini i sumi računa putem *AJAX* poziva. Stoga će nakon obavljenog prebacivanja dostupnih grupa na drugu repliku, aplikacija odmah izvršiti upit na novoj primarnoj replici te će naziv te replike biti prikazan na početnoj stranici.

Prijava u sustav

Kako bi se aplikacija mogla koristiti, potrebno je obaviti prijavu u sustav (Slika 6.7).



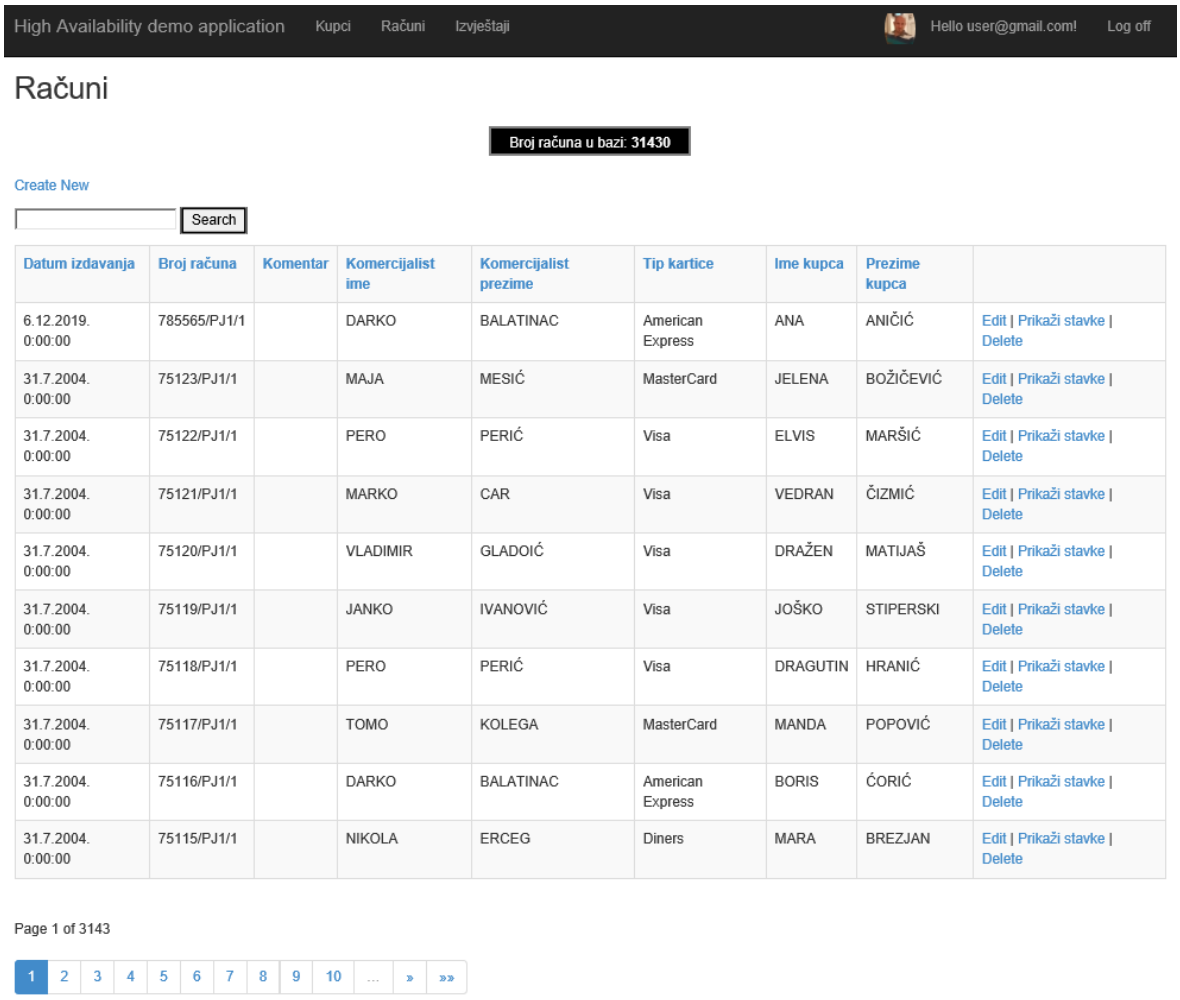
Slika 6.7 Prijava u sustav i registracija

Ako korisnik ne posjeduje korisničko ime i lozinku za prijavu, mora prvo obaviti registraciju klikom na poveznicu za registraciju.

Nakon uspješne prijave aplikacija preusmjerava korisnika na početnu stranicu. Svaki prijavljeni korisnik može ući u izbornike "Račun" i "Kupac", dok za ulazak u izbornik "Izvještaji", treba imati administratorske ovlasti. Ako neprijavljeni korisnik pokuša ući u neki od izbornika, aplikacija će ga preusmjeriti na stranicu za prijavu.

Računi

Prilikom ulaska u izbornik "Računi", prikazuje se lista svih računa sortiranih silazno po broju računa (Slika 6.8).

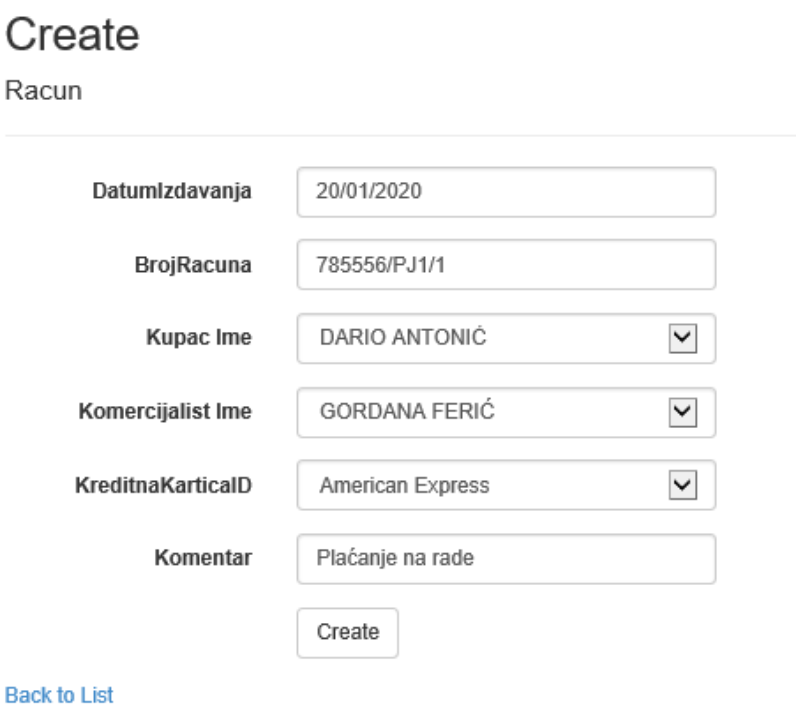


Slika 6.8 Prikaz izbornika "Računi"

Račune je moguće sortirati ulazno i silazno po svim stupcima tablice. Zbog velike količine računa koji se nalazi u bazi (oko 31000), na stranici za prikaz računa implementirano je straničenje (engl. *paging*) koja prikazuje 10 redaka po stranici, kako učitavanje stranice ne bi bilo usporeno.

Za svaki račun moguće je prikazati stavke, urediti ga ili izbrisati, ali pravo na mijenjanje podatka i brisanje računa imaju samo korisnici s administratorskim ovlastima. Računi se mogu pretraživati po svim stupcima upisom željenog teksta te pritiskom na gumb *Search*.

Na ovoj stranici se može kreirati račun odabirom poveznice za kreiranje računa (Slika 6.9).



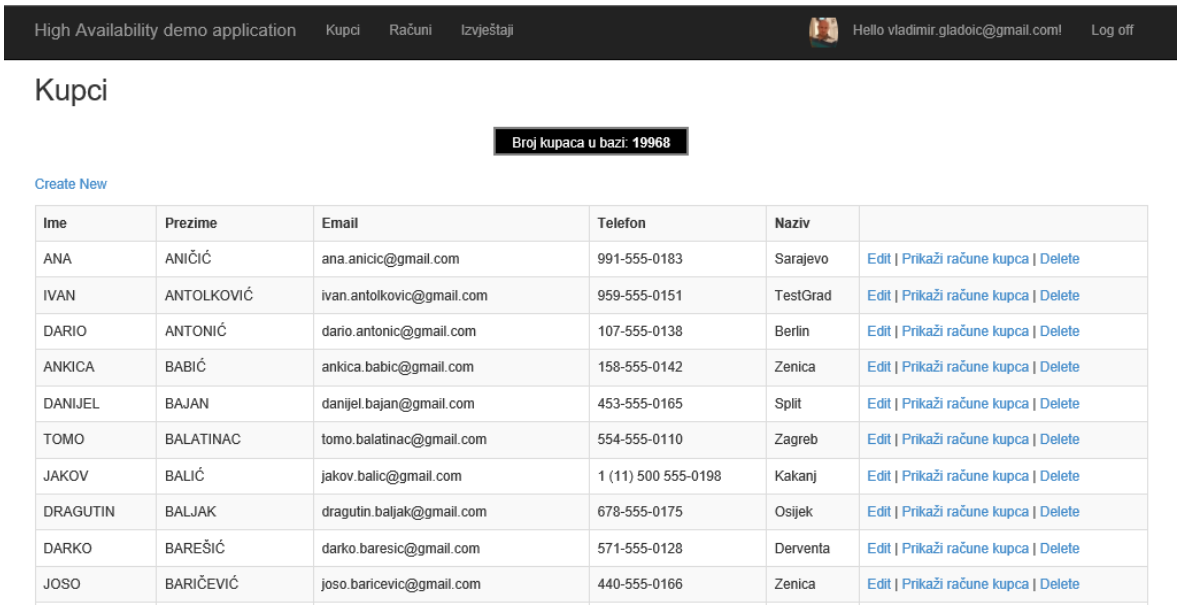
Slika 6.9 Kreiranje računa

Na svim poljima je implementirana validacija pomoću *Data Annotations* atributa. Radi se o ugrađenim validacijskim atributima za različita validacijska pravila. *ASP.NET* okvir automatski primjenjuje validacijska pravila te prikazuje validacijsku poruku u pogledu.[53]

Kupci

Pri ulasku u izbornik "Kupci", prikazuje se lista svih kupaca, kao što je prikazano na slici (Slika 6.10). Svakog kupca moguće je urediti, izbrisati te prikazati samo njegove račune.

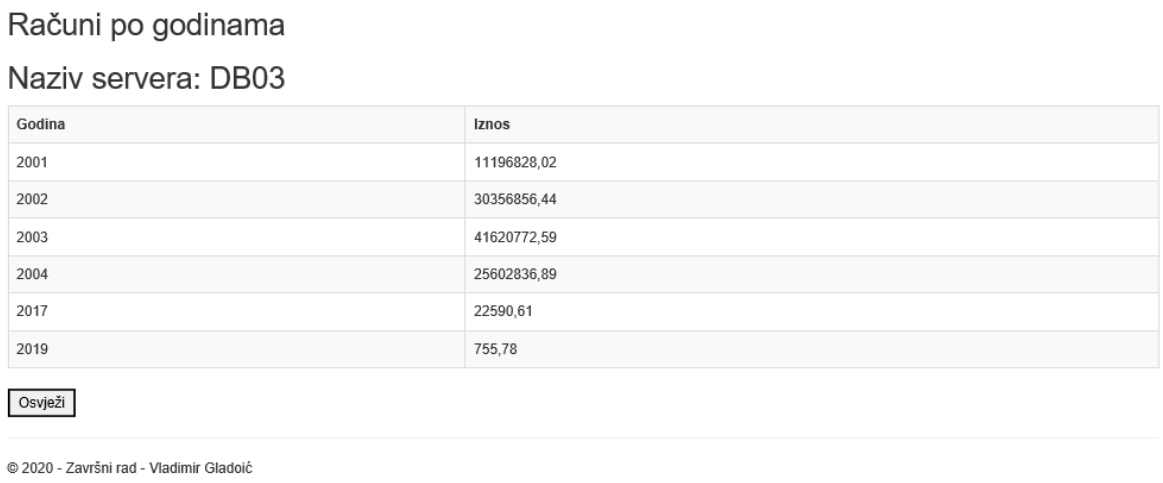
[53] https://www.tutorialsteacher.com/mvc/implement-validation-in-asp.net-mvc; 21.01.2020.



Slika 6.10 Prikaz kupaca

Izvještaji

Izbornik "Izvještaji" kreiran je za potrebe demonstracije rada s replikama za čitanje (Slika 6.11).



Slika 6.11 Prikaz izvještaja

Prikazuje zbirni izvještaj iznosa računa po godina, a kreira poziv koristeći *AG listener* naziv, kao i pozivi iz ostalih izbornika, uz razliku da ima dodano svojstvo `ApplicationIntent=ReadOnly`. To znači da je preusmjeren isključivo na replike za čitanje, a *AlwaysOn* sustav vodi brigu na koju će ga od replika za čitanje preusmjeriti, ovisno o tome kako je konfigurirana distribucija opterećenja u postavkama *AlwaysOn* sustava. Iako je primarna replika trenutno DB01 poslužitelj, ovaj izvještaj je prikazao podatke dohvaćene s DB03 poslužitelja jer je konfiguriran kao replika za čitanje.

7. Analiza performansi i latencija

Kako bi se provela analiza učinka dodavanja sekundarnih replika u dostupne grupe, korištene su *T-SQL* skripte te *Performance Monitor* ugrađen u operativni sustav. Osim ovog softvera, korišten je i softver za simulaciju produkcijskog opterećenja pod nazivom *SQL Query Stress*. Pomoću ovog softvera simulirani su produkcijski uvjeti jer omogućuje istovremeno otvaranje više konekcija prema *SQL Serveru* te može izvršavati naredbe u više dretvi i iteracija. Također, omogućava pokretanje uskladištenih procedura s dinamičkom izmjenom ulaznih parametara, kako bi se uspješno moglo provesti testiranje s podacima s diska i iz predmemorije.

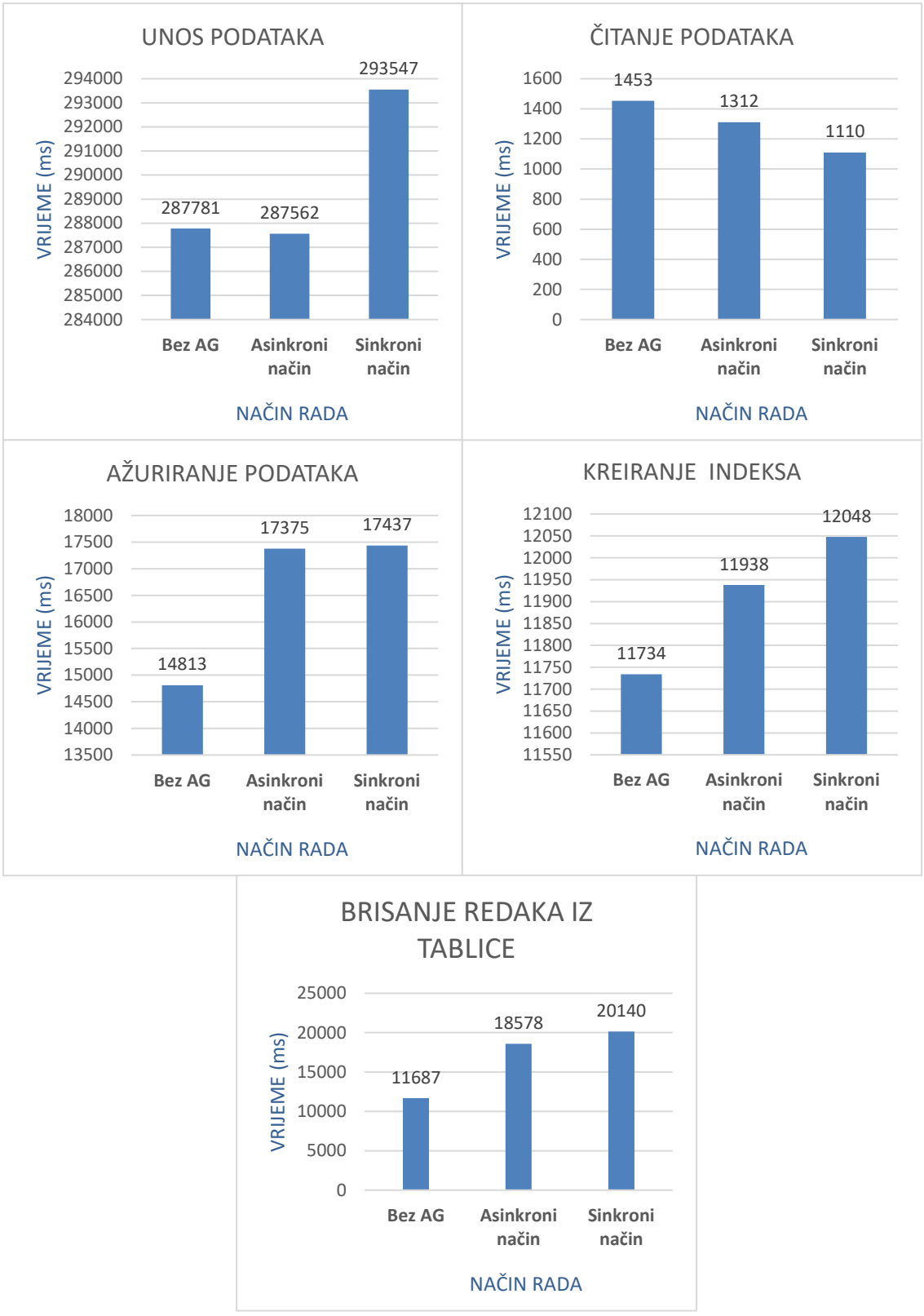
7.1. Utjecaj dodavanja sekundarnih replika na performanse *CRUD* operacija

Za potrebe ovog rada izvršit će se različita mjerenja performansi sustava prilikom izvršavanja *CRUD* operacija prije dodavanja baze u dostupnu grupu, nakon dodavanja u dostupnu grupu u asinkronom načinu rada te nakon promjene u sinkroni način rada. Nakon tako prikupljenih podataka bit će napravljen usporedni prikaz rezultata sva tri mjerenja te će biti izveden zaključak o utjecaju dodavanja sinkronih i asinkronih replika na *CRUD* operacije.

Kako bi se izvršila potrebna mjerenja, kreirana je nova baza pod nazivom ZavrsniRad_Mjerenja te su kreirane potrebne tablice i popunjene skriptno generiranim testnim podacima. Skripte kojima je kreirana baza te njeni objekti bit će priložene uz završni rad. Kako bi se istražilo kakav utjecaj dodavanje sekundarnih replika ima na performanse *CRUD* operacija, mjereno je proteklo vrijeme (engl. *elapsed time*) te procesorsko vrijeme (engl. *CPU time*) za sljedeće operacije na uzorku od 1,000.000 redaka:

- unos redaka u tablicu
- čitanje redaka iz tablice
- ažuriranje redaka u tablici
- brisanje redaka iz tablice
- kreiranje neklasteriranog indeksa.

Provedenim mjerenjima došlo se do rezultata prikazanih u tablici (**Error! Reference source not found.**).



Slika 7.1 Rezultati mjerenja performansi CRUD operacija[54]

[54] Izvor: Vlastiti rad autora; 3.1.2020.

Na temelju dobivenih rezultata, doneseni su sljedeći zaključci:

1. **Kod unosa podataka**
    - dodavanje sekundarne replike u asinkronom načinu ne utječe značajno na performanse te su razlike neprimjetne u odnosu na unos podatka u bazu bez sekundarnih replika.
    - dodavanje sekundarne replike u sinkronom načinu rada ima utjecaj na performanse te je unos sporiji za 2.00 % u odnosu na unos bez sekundarne replike.
2. **Kod čitanja podataka**
    - dodavanje sekundarne replike, bilo u sinkronom ili asinkronom načinu rada, nema utjecaja na performanse te su minimalne razlike dobivene mjerenjima posljedica trenutnog zauzeća sustava.
3. **Kod ažuriranja podataka**
    - dodavanje replike u sinkronom ili asinkronom načinu rada ima podjednak utjecaj na performanse te je ažuriranje sporije za 17.71 % u odnosu na ažuriranje bez sekundarne replike.
4. **Kod kreiranja indeksa**
    - dodavanje sekundarne replike u asinkronom načinu rada utječe na performanse te je kreiranje indeksa sporije za 1.74 % u odnosu na kreiranje indeksa kada nema sekundarne replike.
    - dodavanje replike u sinkronom načinu još više utječe na performanse pa je, u odnosu na kreiranje indeksa u bazi koja nema sekundarnu repliku, sporije za 2.67 %.

U ovom poglavlju se analizirao utjecaja dodavanja sekundarnih replika na pojedine *CRUD* operacije. Mjerenje utjecaja na performanse sustava u kojem je simulirano produkcijsko opterećenje opisuje poglavlje 7.6. (Rezultati mjerenja).

7.2. Utjecaj dodavanja replike za čitanje na performanse

Sekundarne replike za čitanje u pravilu rasterećuju primarnu repliku od *read-only* upita, što pozitivno utječe na performanse primarne replike prilikom produkcijskog opterećenja, ali u nekim slučajevima može i negativno utjecati na performanse primarne replike. Ako se radi o sekundarnoj replici koja je u asinkronom načinu rada, nema utjecaja na transakcijski

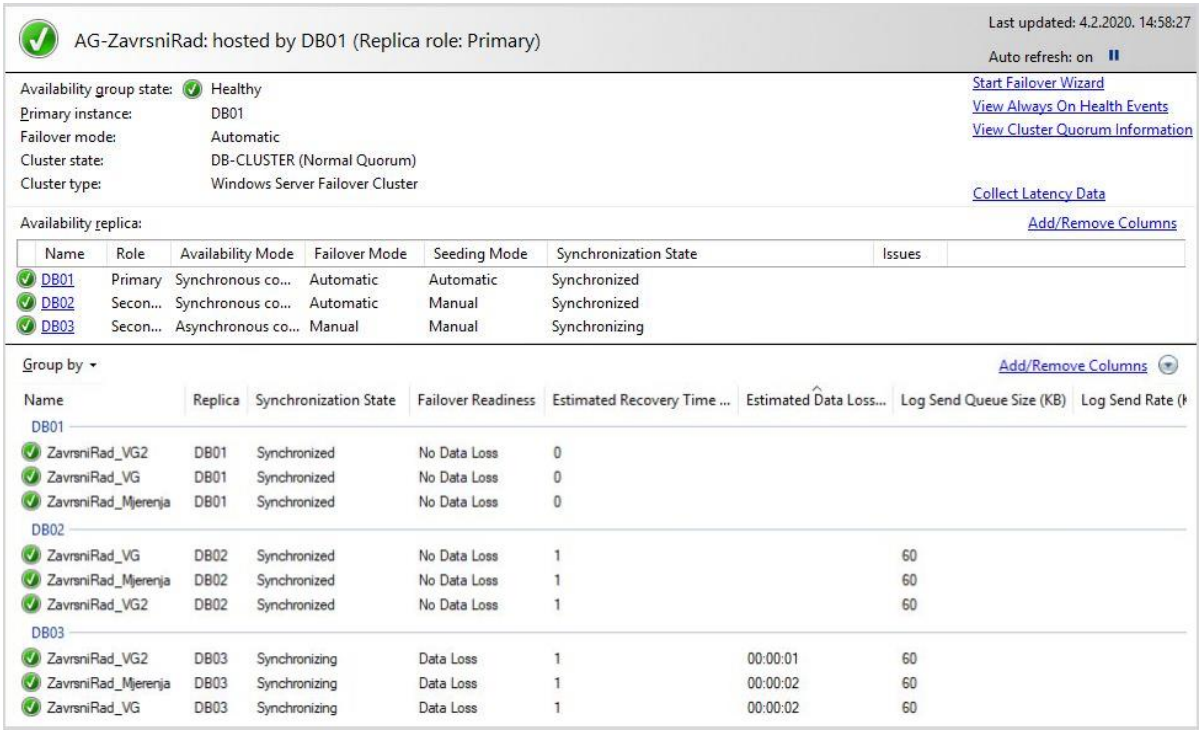
odgovor na primarnoj replici. Međutim, ako je sekundarna replika konfigurirana za sinkroni način rada, onda primarna replika izvršava transakcije tek kada je zapis s naredbom za izvršenje (engl. *commit*) zapisan na sekundarnoj replici.[55] Tek tada sekundarna replika šalje potvrdu primarnoj da je transakcija izvršena. Upravo to čekanje na potvrdu sekundarne replike uzrokuje latenciju pri izvršenju transakcija. Latencija se u tom slučaju dodatno povećava ako postoje problemi na mrežnoj razini. Drugi način na koji sekundarna replika može utjecati na opterećenje primarne replike je tako da se uslijed velikog opterećenja upitima za čitanje na sekundarnoj replici uspori slanje potvrde o uspješno izvršenoj transakciji. To je moguće samo u rijetkim situacijama jer *SQL Server* daje prioritet pozadinskim nitima (engl. *background threads*) naspram korisničkim. Tako dretve za primanje zapisa i zapisivanje istih u zapisničku datoteku imaju prioritet u odnosu na korisničke operacije čitanja. Stoga, iz perspektive procesora, takve operacije čitanja ne bi trebale odgađati potvrdu izvršenja transakcije. Iz perspektive sustava za pohranu podataka, intenzivne operacije čitanja mogu usporiti zapisivanje transakcijskih zapisa, ako podatkovna i zapisnička datoteka dijele isti fizički sustav za pohranu podataka.[56]

7.3. Monitoring performansi *AlwaysOn* dostupnih grupa

Za nadgledanje *AlwaysOn* dostupnih grupa, na raspolaganju stoje *AlwaysOn Dashboard* implementiran unutar *SQL Servera* ili neki od vanjskih alata kao što je *System Operations Center* ili *SentryOne*. Osim gotovih rješenja, moguće je kreirati *Extended Events* sesije za nadzor dostupnih grupa ili upite koristeći poglede *Dynamic Management View* (skraćeno DMV).

AlwaysOn Dashboard predstavlja interaktivni izvještaj koji u realnom vremenu prikazuje stanje *AlwaysOn* okruženja te omogućuje pregledavanja elemenata *AlwaysOn* topologije (Slika 7.2).

[55] AlwaysOn: Offloading Read-Only Workloads to Secondary Replicas; Sunil Agarwal; 20.1.2020.
[56] Ibidem;20.1.2020.



Slika 7.2 *AlwaysOn dashboard*

Kao što se vidi na slici (Slika 7.2), unutar *AlwaysOn Dashboarda* u gornjem dijelu prikazan je naziv dostupnih grupa uključujući i njihovo stanje. Za svaku dostupnu grupu se može vidjeti podatak o stanju klastera, nazivu primarnog poslužitelja u AG te informaciju o tome radi li u automatskom ili ručnom načinu prebacivanja na sekundarnu instancu. Ispod su prikazane replike koje sudjeluju u dostupnoj grupi te načini rada svake od njih. Osim toga, prikazuje i nazive svih baza koje sudjeluju u dostupnim grupama te stanja svake od njih. Tako se za svaku bazu može vidjeti je li sinkronizirana s primarnim poslužiteljem, koliki je trenutni RTO i RPO, kolike su brzine prijenosa i primjene zapisa na sekundarne replike i podatak o tome kada je poslana i zaprimljena zadnja transakcija na sekundarnoj replici.

AlwaysOn sustav se može nadgledati koristeći *Extended Event* sesije te njihove zapise, koji mogu pružiti informacije o promjenama stanja dostupne grupe ili baza, javljati greške te informacije o *Data Definition Language* (skraćeno DDL) aktivnostima koje se odnose na dostupne grupe.

7.4. Procjena vremena potrebnog za *failover* (RTO)

Sa stajališta aplikacije, vrijeme potrebno za prebacivanje na sekundarnu repliku (RTO) predstavlja maksimalnu količinu vremena koje aplikacija može biti nedostupna prije nego

se baza podataka oporavi te je korisnici mogu ponovo početi koristiti.[57] Koliko brzo će se baza oporaviti nakon prebacivanja na sekundarnu repliku, ovisi o mnogim faktorima, poput veličine baza, količine baza na instanci te o broju transakcija koje se trenutno izvršavaju. Ako se prebacivanje na sekundarnu repliku dogodi u trenutku izvršenja transakcija, sve neizvršene transakcije moraju se poništiti (engl. *rollback*) pa to povećava vrijeme potrebno za oporavak.Vrijeme potrebno za prebacivanje baza na sekundarnu repliku (RTO), koje se definira u ugovoru o razini dostupnosti, ovisi o vremenu unutar kojeg se može odviti proces prebacivanja baza konkretnog *AlwaysOn* sustava, a može se izraziti sljedećom formulom (1):

$$T_{prebacivanja} = T_{detektiranja} + T_{čitanja\ zapisa} + T_{prebacivanja}[58] \quad (1)$$

Ovdje $T_{detektiranja}$ predstavlja vrijeme potrebno da sustav detektira kvar, a $T_{čitanja\ zapisa}$ vrijeme potrebno da *redo dretva* dođe do kraja čitanja zapisa (engl. *log*). $T_{prebacivanja}$ predstavlja vrijeme potrebno klasteru za prebacivanje i oporavak baza na sekundarnu repliku. Ako se dostupna grupa sastoji od više baza, tada baza s najvećim vremenom prebacivanja ($T_{prebacivanja}$) postaje granična vrijednost za usklađivanje s definiranim RTO[59]. Kako bi se izračunalo vrijeme potrebno za prebacivanje baza na sekundarnu repliku, najprije je potrebno izračunati vrijednost $T_{primjene}$ prema formuli (2):

$$T_{primjene} = \text{količina zapisa koji čeka izvršenje / količina zapisa koji se izvršava u sekundi}[60] \quad (2)$$

Kako bi se simuliralo stanje u kojem baze sekundarnih replika (DB02 i DB03) nisu sinkronizirane s primarnom replikom, pokrenut je upit s intenzivnim *CRUD* operacijama te zatim u drugom krugu mjerenja, upit koji simulira redovno produkcijsko opterećenje. Tako su stvoreni preduvjeti za mjerenje RTO i RPO vrijednosti pri redovnom produkcijskom opterećenju i prilikom izvođenja zahtjevnih *CRUD* operacija. Primijenivši formulu za izračun vremena potrebnog za prebacivanje baza na sekundarnu repliku (2) u *T-SQL* skripti, moglo se izračunati koliki je RTO za svaku pojedinu repliku i bazu koje sudjeluje u dostupnoj grupi (Programski isječak 7.1).

[57] SQL Server AlwaysOn Revealed, 2nd Edition; P. A. Carter; str. 4.
[58] https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2012/dn135338(v=sql.110)?redirectedfrom=MSDN; 20.01.2020.
[59] https://www.sqlshack.com/measuring-availability-group-synchronization-lag/; 20.1.2020.
[60] Ibidem; 20.1.2020.

```
;WITH
AG_Stats AS
(SELECT AR.replica_server_name,
        HARS.role_desc,
        Db_name(DRS.database_id) [DBName],
        DRS.redo_queue_size redo_queue_size_KB,
        DRS.redo_rate redo_rate_KB_Sec
        FROM   sys.dm_hadr_database_replica_states DRS
            INNER JOIN sys.availability_replicas AR
                  ON DRS.replica_id = AR.replica_id
            INNER JOIN sys.dm_hadr_availability_replica_states HARS
      ON AR.group_id = HARS.group_id
            AND AR.replica_id = HARS.replica_id),
Pri_CommitTime AS
(SELECT     replica_server_name, DBName
            , redo_queue_size_KB
            , redo_rate_KB_Sec
  FROM      AG_Stats
  WHERE     role_desc = 'PRIMARY'),
Sec_CommitTime AS
(SELECT     replica_server_name
            , DBName
            , redo_queue_size_KB
            , redo_rate_KB_Sec
FROM  AG_Stats WHERE    role_desc = 'SECONDARY')
SELECT p.replica_server_name [primary_replica]
      , p.[DBName] AS [DatabaseName]
      , s.replica_server_name [secondary_replica]
      ,s.redo_queue_size_KB
      ,s.redo_rate_KB_Sec
      ,CONVERT(decimal(18,4),ROUND(convert(decimal(18,5),
      s.redo_queue_size_KB)
      / convert(decimal(18,5),s.redo_rate_KB_Sec),4)) as
      [Redo_Lag_Secs]
      FROM Pri_CommitTime p
      LEFT JOIN Sec_CommitTime s ON [s].[DBName] = [p].[DBName]
```

Programski isječak 7.1 *T-SQL* skripta za izračun RTO[61]

[61] https://www.sqlshack.com/measuring-availability-group-synchronization-lag/; 20.01.2020.

Time su dobiveni rezultati koji pokazuju kolika su kašnjenja pri izvršenju transakcija (*Redo lag*) na sekundarnim replikama u odnosu na primarnu repliku, prilikom redovnog produkcijskog opterećenja (Tablica 7.1) te prilikom izvršavanja intenzivnih *CRUD* operacija (

Tablica 7.2). Kako bi se prema formuli za izračun RTO (1) izračunala vrijednost RTO, tom kašnjenju dodaje se još i vrijeme potrebno da sustav detektira kvar (otprilike 10 sec) te vrijeme koje je potrebno klasteru za prebacivanje i oporavak baza na sekundarnoj replici (otprilike 5 sec).

Tablica 7.1 Kašnjenja u izvršavanju transakcija prilikom redovnog produkcijskog opterećenja[62]

Primary replica	Database name	Secondary replica	Redo queue (KB)	Redo rate (KB/Sec)	Redo lag (sec)
DB01	ZavrsniRad_VG	DB02	28	16182	0,0017
DB01	ZavrsniRad_VG	DB03	56	15359	0,0036
DB01	ZavrsniRad_VG2	DB02	28	23479	0,0012
DB01	ZavrsniRad_VG2	DB03	0	21735	0,0000
DB01	ZavrsniRad_Mjerenja	DB02	28	32204	0,0009
DB01	ZavrsniRad_Mjerenja	DB03	56	30585	0,0018

Tablica 7.2 Kašnjenja u izvršavanju transakcija prilikom izvođenja intenzivnih *CRUD* operacija[63]

Primary replica	Database name	Secondary replica	Redo queue (KB)	Redo rate (KB/Sec)	Redo lag (sec)
DB01	ZavrsniRad_VG	DB02	60	18139	0,0033
DB01	ZavrsniRad_VG	DB03	120	13296	0,0090
DB01	ZavrsniRad_VG2	DB02	360	16646	0,0216
DB01	ZavrsniRad_VG2	DB03	420	13921	0,0302
DB01	ZavrsniRad_Mjerenja	DB02	360	28160	0,0128
DB01	ZavrsniRad_Mjerenja	DB03	360	27168	0,0133

[62] Izvor: Vlastiti rad autora; 22.01.2020.
[63] Izvor: Vlastiti rad autora; 22.01.2020.

Na ovom primjeru vidi se da je kašnjenje pri zapisivanju podataka u bazu na sekundarnu repliku u svim slučajevima gotovo jednako nuli, ali zbog vremena potrebnog za detektiranje kvara i vremena potrebnog za prebacivanje na sekundarnu repliku, ipak RTO ovog sustava iznosi otprilike 8 sekundi.

7.5. Procjena potencijalnog gubitka podataka (RPO)

Mogući gubitak podatka izražava se u jedinici vremena pa se tako npr. može reći da je RPO nekog visoko dostupnog sustava 60 sekundi. RPO implementiranog *AlwaysOn* sustava moguće je izračunati primjenom sljedeće formule (3):

$$\text{Vrijeme}_{\text{gubitka podataka}} = \text{vrijeme izvršenja zadnje transakcije (primarni poslužitelj) - vrijeme izvršenja zadnje transakcije (sekundarni poslužitelj)} \quad (3)$$

Navedena formula primijenjena je u T-SQL skripti za izračun RPO vrijednosti (Programski isječak 7.2):

```
;WITH
AG_Stats AS
(SELECT AR.replica_server_name,
       HARS.role_desc,
       Db_name(DRS.database_id) [DBName],
       DRS.last_commit_time
       FROM   sys.dm_hadr_database_replica_states DRS
       INNER JOIN sys.availability_replicas AR
            ON DRS.replica_id = AR.replica_id
       INNER JOIN sys.dm_hadr_availability_replica_states HARS
            ON AR.group_id = HARS.group_id
            AND AR.replica_id = HARS.replica_id),
Pri_CommitTime AS
(SELECT replica_server_name, DBName
       , last_commit_time
FROM AG_Stats
WHERE role_desc = 'PRIMARY'),
Sec_CommitTime AS
(SELECT replica_server_name
       , DBName
       , last_commit_time
       FROM AG_Stats WHERE role_desc = 'SECONDARY')
SELECT p.replica_server_name [primary_replica]
```

```
        , p.[DBName] AS [DatabaseName]
        , s.replica_server_name [secondary_replica]
        , DATEDIFF(ss,s.last_commit_time,p.last_commit_time) AS
      [Sync_Lag_Secs]
FROM Pri_CommitTime p
      LEFT JOIN Sec_CommitTime s
            ON [s].[DBName] = [p].[DBName]
```

Programski isječak 7.2 T-SQL skripta za izračuna RPO[64]

Time se došlo do rezultata koji prikazuju koliki je stvarni RPO prilikom redovitog produkcijskog opterećenja (Tablica 7.3) te prilikom izvođenja intenzivnih *CRUD* operacija (Tablica 7.4).

Tablica 7.3 RPO prilikom redovnog produkcijskog opterećenja[65]

Primary replica	Database name	Secondary replica	Last commit time-primary	Last commit time - secondary	Sync Lag (sec)
DB01	ZavrsniRad_VG	DB02	29.1.2020 10:47	29.1.2020 10:47	1
DB01	ZavrsniRad_VG	DB03	29.1.2020 10:47	29.1.2020 10:47	1
DB01	ZavrsniRad_VG2	DB02	29.1.2020 10:47	29.1.2020 10:47	0
DB01	ZavrsniRad_VG2	DB03	29.1.2020 10:47	29.1.2020 10:47	0
DB01	ZavrsniRad_Mjerenja	DB02	29.1.2020 10:47	29.1.2020 10:47	0
DB01	ZavrsniRad_Mjerenja	DB03	29.1.2020 10:47	29.1.2020 10:47	0

Tablica 7.4 RPO prilikom izvođenja intenzivnih *CRUD* operacija [66]

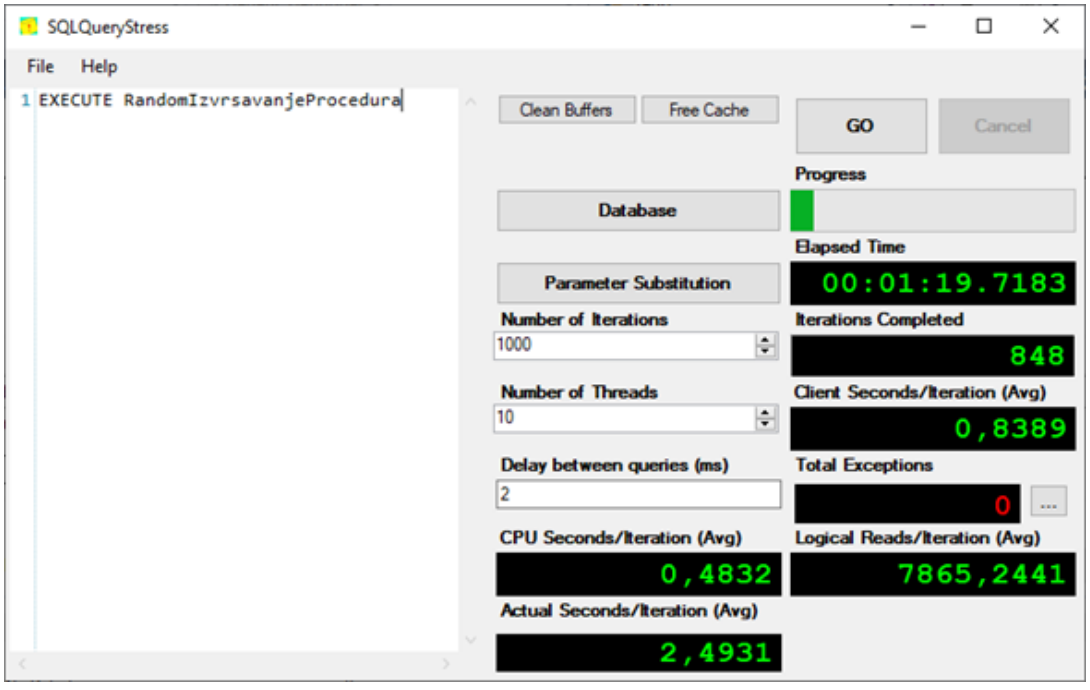
Primary replica	Database name	Secondary replica	Last commit time - primary	Last commit time - secondary	Sync Lag (sec)
DB01	ZavrsniRad_VG	DB02	2020-01-29 10:19:05.700	2020-01-29 10:19:07.180	2
DB01	ZavrsniRad_VG	DB03	2020-01-29 10:19:05.573	2020-01-29 10:19:07.180	2
DB01	ZavrsniRad_VG2	DB02	2020-01-29 10:19:06.303	2020-01-29 10:19:07.470	1
DB01	ZavrsniRad_VG2	DB03	2020-01-29 10:19:06.597	2020-01-29 10:19:07.470	1
DB01	ZavrsniRad_Mjerenja	DB02	2020-01-29 10:19:06.630	2020-01-29 10:19:07.753	1
DB01	ZavrsniRad_Mjerenja	DB03	2020-01-29 10:19:06.290	2020-01-29 10:19:07.753	1

[64] https://www.sqlshack.com/measuring-availability-group-synchronization-lag/; 21.01.2020.
[65] Izvor: Vlastiti rad autora; 22.01.2020.
[66] Ibidem; 22.01.2020.

Razlika u sinkronizaciji (engl. *Sync lag*) u ovom slučaju predstavlja RPO izražen u sekundama. Kao što se vidi iz tablice, RPO se neznatno razlikuje ako usporedimo redovno produkcijsko opterećenje s opterećenjem prilikom izvršavanja intenzivnih *CRUD* operacija. Prilikom redovnog produkcijskog opterećenja RPO se kreće između 0 i 1 sekunde, dok se prilikom izvršavanja intenzivnih *CRUD* operacija, poput kreiranja indeksa, unosa ili ažuriranja milijun zapisa, RPO kreće između 1 i 2 sekunde. To znači da se može izgubiti maksimalno toliko podataka izraženo u vremenskoj jedinici. Iako Microsoft jamči da je kod sinkrone replike RTO = 0, tj. da nije moguć gubitak podataka, ovdje se vidi da postoji RPO od nekoliko sekundi. To pokazuje da je RPO = 0 samo u idealnim uvjetima, jer se transakcijski zapisi o izvršenim transakcijama zapisuju u zapisničku datoteku prije slanja potvrde na primarni poslužitelj, a to ne jamči da će *redo thread* na sekundarnoj replici doista i primijeniti pripadajuće zapisničke retke u podatkovne stranice.[67]

7.6. Rezultati mjerenja

Kako bi se istražio utjecaj dodavanja sekundarnih replika na performanse sustava, simulirano je produkcijsko opterećenje korištenjem softvera *SQL Query Stress* (Slika 7.3).



Slika 7.3 *SQL Query Stress* - softver za simuliranje opterećenja *SQL Servera*

[67] AlwaysOn: Offloading Read-Only Workloads to Secondary Replicas; Sunil Agarwal; 23.1.2020.

Pomoću njega je pokrenuto nasumično izvršavanje uskladištenih procedura za čitanje, pisanje, ažuriranje te brisanje podataka s dinamičkim mijenjanjem ulaznih parametara. Time se postiglo da uskladištene procedure ne dohvaćaju podatke samo iz predmemorije, već i s diska, kako bi se stvorilo okruženje što sličnije produkcijskom. Dodatno, korištena je uskladištena procedura koja nasumično poziva druge uskladištene procedure (Programski isječak 7.3).

```
CREATE PROCEDURE [dbo].[RandomIzvrsavanjeProcedura] WITH
RECOMPILE
AS
SET NOCOUNT ON
DECLARE @Id INT
SELECT @Id = CAST(RAND() * 100000 AS INT)
IF @Id % 5 = 0
    EXEC dbo.IzvrsiUnos @Id
ELSE IF @Id % 4 = 0
    EXEC dbo.IzvrsiAzuriranje @Id
ELSE IF @Id % 3 = 0
      EXEC dbo.IzvrsiDohvat @Id
ELSE IF @Id % 2 = 0
    EXEC dbo.IzvrsiBrisanje @Id
ELSE
    EXEC dbo.IzvrsiDohvat2 @Id
      END
GO
```

Programski isječak 7.3 Uskladištena procedura za pozivanje drugih procedura

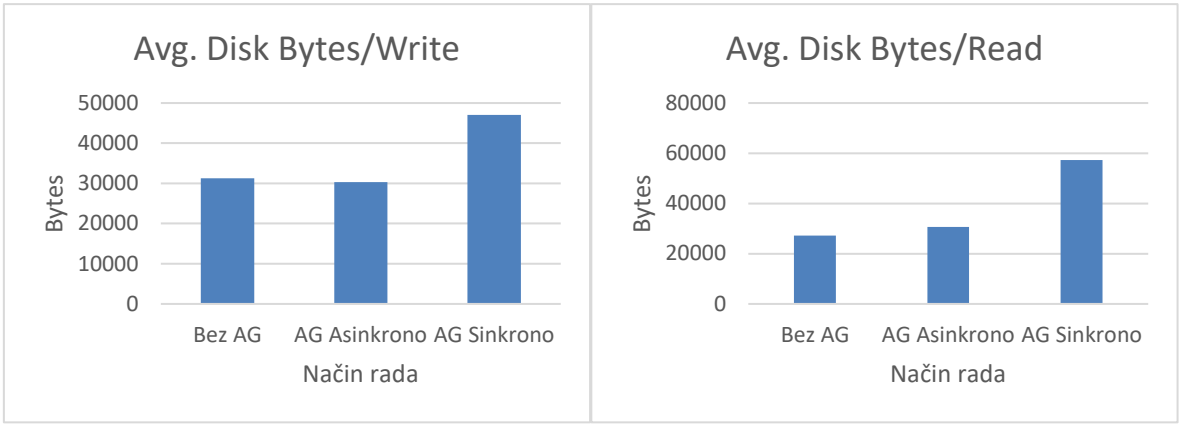
Unutar *SQL Query Stress* softvera odabrano je da se pozivanje uskladištene procedure `[dbo].[RandomIzvrsavanjeProcedura]`izvršava u 1000 iteracija te u 10 dretvi, kako bi se simuliralo istovremeno korištenje baze podatka od strane 10 korisnika. Tako je stvoreno opterećenje na *SQL Server* u prosječnom trajanju od 10 minuta dok baze nisu bile dodane u AG, koje je onda ponovljeno s bazama u AG u sinkronom te asinkronom načinu rada. U svrhu snimanja te kasnije usporedbe izmjerenih vrijednosti, korišten je softver *Performance Monitor* koji je integriran u operativni sustav. Pomoću njega su mjereni različiti brojači performansi (engl. *performance counters*), kako je prikazano u tablici (Tablica 7.5)

Tablica 7.5 Brojači performansi

Brojač performansi	Kategorija	Opis
Avg. Disk Bytes/Write	Logical Disk	Prikazuje prosječnu veličinu *write* operacija na disku izraženu u bajtovima
Avg. Disk Bytes/Read	Logical Disk	Prikazuje prosječnu veličinu *read* operacija na disku izraženu u bajtovima u sekundi
Bytes Recieved/sec	Network Interface	Prikazuje količinu primljenog prometa preko mrežnog adaptera izraženu u bajtovima u sekundi
Bytes Sent/sec	Network Interface	Prikazuje količinu poslanog prometa preko mrežnog adaptera izraženu u bajtovima
Packets Recieved/sec	Network Interface	Prikazuje količinu zaprimljenih paketa preko mrežnog sučelja u sekundi
Packets Sent/sec	Network Interface	Prikazuje količinu poslanih paketa preko mrežnog sučelja u sekundi
%DPC Time	Processor	Predstavlja vrijeme koje pojedini procesor provede u zaprimanju i služenju odgođenih proceduralnih poziva
%Privileged Time	Processor	Predstavlja vrijeme koje procesor utroši izvršavajući sistemske pozive
Batch Requests/sec	SQL	Prikazuje broj zaprimljenih skupnih T-SQL naredbi po sekundi
Transactions/sec	SQL	Prikazuje broj izvršenih transakcija u sekundi
Log Writes waits/ms	SQL	Predstavlja broj čekanja prilikom pisanje u međuspremnik zapisa (engl. *log buffer*) u milisekundi
Network IO waits/ms	SQL	Prikazuje broj čekanja na razini mreže u milisekundi

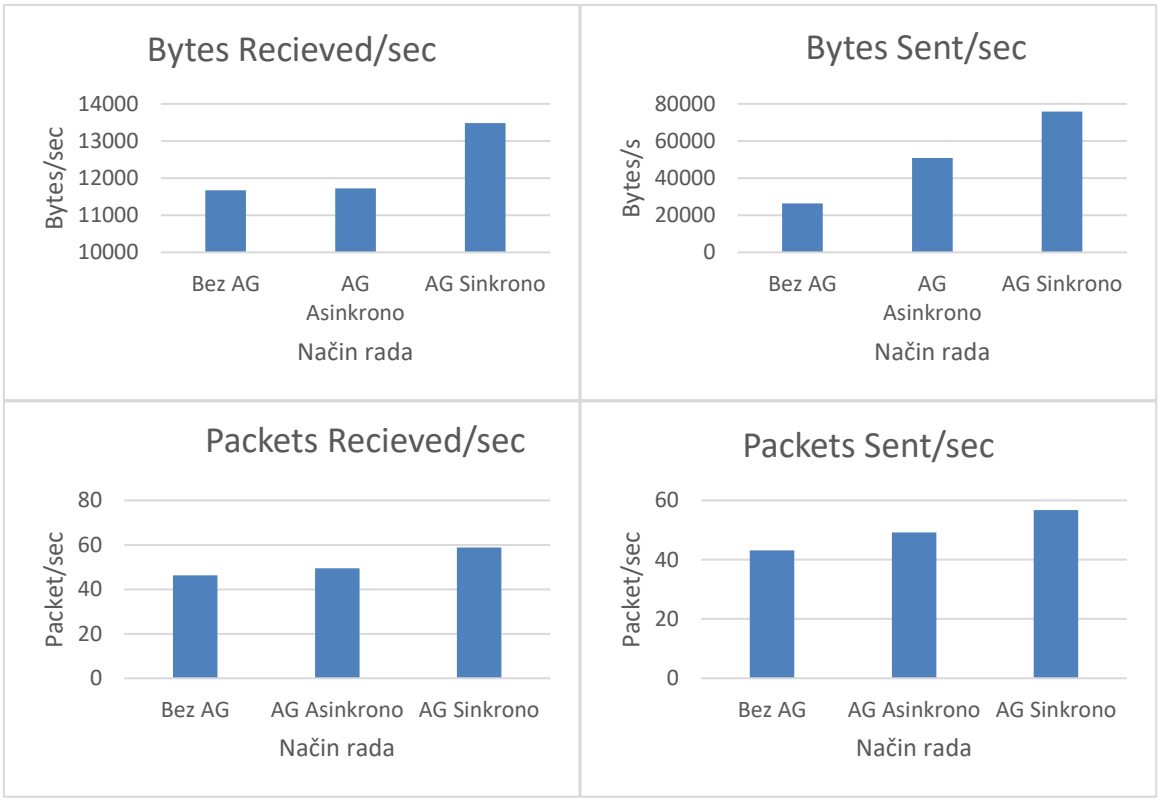
Mjerenjem navedenih brojača prikupljeni su rezultati koje će biti grafički prikazani, radi lakšeg uočavanja razlika u usporednom prikazu vrijednosti prije dodavanja u AG i nakon dodavanja baza u AG u sinkronom te asinkronom načinu rada.

Unutar kategorije "*Logical Disk*" nalaze se brojači performansi pod nazivom *Avg. Disk Bytes/Write* i *Avg. Disk Bytes/Read* koji nam pokazuju prosječnu veličinu pisanja, odnosno čitanja po disku. U grafičkom prikazu (Slika 7.4) uočava se da dodavanje sekundarne replike u asinkronom načinu rada nema skoro nikakav utjecaj na veličinu operacija pisanja, dok dodavanje replike u sinkronom načinu rada pokazuje značajan utjecaj na prosječnu veličinu pojedine operacije pisanja.



Slika 7.4 Brojači performansi iz kategorije "*Logical Disk*"

Prosječna veličina pojedine operacije pisanja poveća se za 50.31 % u sinkronom načinu rada u odnosu na rad s bazom koja nije u dostupnoj grupi. Prosječna veličina operacija čitanja nakon dodavanja sekundarne replike u asinkronom načinu rada poraste za 12.62 % u odnosu na operacije na bazi koja nije u dostupnoj grupi. Ako način rada prebacimo u sinkroni, tada prosječna veličina operacije čitanja poraste za 110 % u odnosu bazu koja nema sekundarnu repliku. Kako bi se izmjerio utjecaj dodavanja sekundarne replike na mrežni promet, izvršena su mjerenja na 4 mrežna brojača performansi: *Bytes Recieved/sec*, *Bytes Send/sec*, *Packet Recieved/sec* i *Packet Sent/sec* (Slika 7.5).

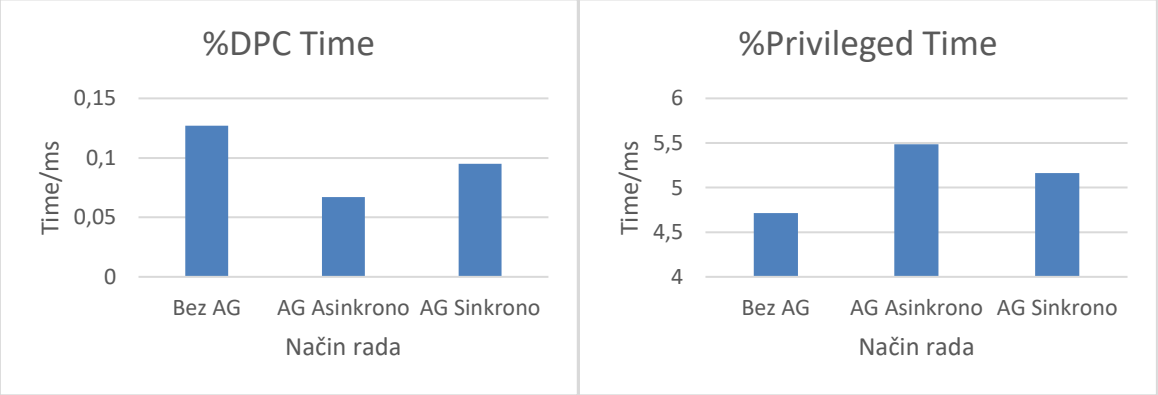


Slika 7.5 Brojači performansi iz kategorije "Network Interface"

Iz dobivenih rezultata svih brojača performansi iz kategorije "*Network Interface*" može se zaključiti da dodavanje sekundarne replike u asinkronom načinu rada utječe na povećanje svih brojača te da promjena sinkronog u asinkroni način rada dodatno utječe na povećanje vrijednosti mjerenih brojača performansi. Tako se prosječna količina primljenih podatka, izraženih u bajtovima u sekundi, povećala za 0.5 % nakon dodavanja sekundarne replike u asinkronom načinu rada, a nakon promjena načina rada u sinkroni za 15 % u odnosu na bazu koja nema sekundarnu repliku. Gleda li se s aspekta poslane količine podataka, vide se veća odstupanja te se tako količina poslanih podataka povećala nakon dodavanja asinkrone replike za 92 %, a nakon promjene u sinkroni način rada za 186 % u odnosu na bazu bez replike.

Što se tiče broja primljenih paketa, prilikom dodavanja asinkrone replike povećanje je od 7 %, odnosno povećanje od 27 % prilikom dodavanja sinkrone replike. Slična je situacija i kod poslanih paketa.

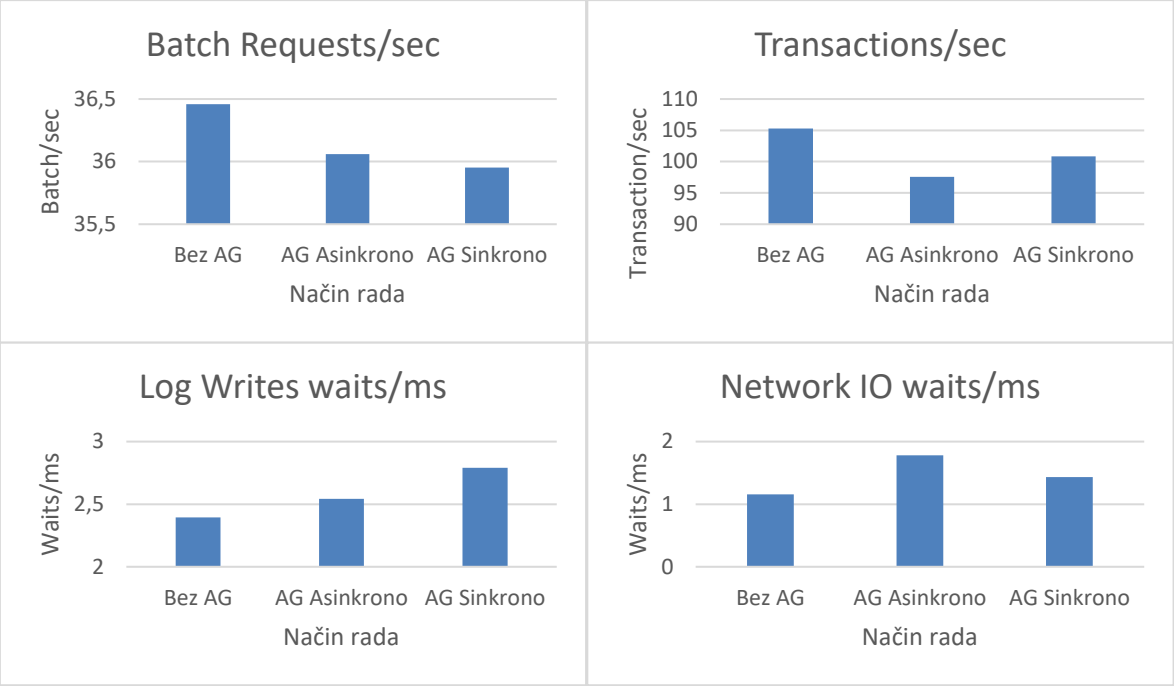
Kako bi se provela analiza zauzeća procesora ovisno o postojanju i načinu rada sekundarnih replika, za mjerenje su odabrana 2 brojača performansi iz kategorije "*Processor*": *%DPC Time* i *%Privileged Time* (Slika 7.6).



Slika 7.6 Brojači performansi iz kategorije "*Processor*"

Kod navedenih brojača performansi nisu uočene nikakve zakonitosti između dodavanja replika u asinkronom ili sinkronom načinu rada te povećanja ili smanjenja zauzeća procesora. Pretpostavka je da su dobiveni rezultati posljedica samo trenutnog zauzeća procesora u trenutku mjerenja, koje može biti uzrokovano bilo kojim drugim procesom koji se trenutno izvršava na poslužitelju.

Unutar kategorije "*SQL*" brojača performansi za analizu su odabrana sljedeća 4 brojača: *Batch Requests/sec*, *Transactions/sec*, *Log Writes waits/ms* te *Network IO waits/ms* (Slika 7.7).



Slika 7.7 Brojači performansi iz kategorije "SQL"

Na temelju izvršenih mjerenja pomoću brojača iz kategorije "*SQL*" doneseni su sljedeći zaključci:

- broj pristiglih i obrađenih skupnih naredbi u sekundi je veći za 1.09 % ako baza nema sekundarnu repliku u odnosu na bazu koja ima asinkronu repliku, a za 1.38 % je veći u odnosu na bazu u sinkronom načinu rada.
- broj transakcija koje se izvrše u sekundi je za 7.3 % veći kada baza nema sekundarnu repliku u odnosu na bazu koja ima asinkronu repliku te je za 4.23 % veći u odnosu na bazu koja ima sinkronu repliku.
- čekanja prilikom zapisivanja u međuspremnik zapisa (engl. *log buffer*) su veća 6.17 % ako postoji sekundarna repliku u asinkronom načinu rada te 16.49 % ako postoji sekundarna repliku u sinkronom načinu rada, u odnosu na bazu bez sekundarne replike.
- čekanja na razini mreže su 53.66 % veća ako postoji replika u asinkronom načinu rada te 23.46 % ako postoji sekundarna replika u sinkronom načinu rada u odnosu na bazu bez sekundarne replike.

Zaključak

Prilikom pisanja ovog rada provedeno je istraživanje i analiza trenutno postojećih rješenja visoke dostupnosti te je za daljnju analizu kao najpotpunije rješenje odabrana tehnologija *AlwaysOn*. Ova tehnologija se pokazala i kao najpouzdanija jer primarna replika može imati više sinkronih replika koje ne dijele zajednički diskovni sustav te se mogu nalaziti na različitim geografskim lokacijama. Provedena je analiza utjecaja dodavanja sekundarnih replika na performanse *CRUD* operacija te je utvrđeno da dodavanje sekundarne replike u sinkronom ili asinkronom načinu rada najviše utječe na operacije ažuriranja podataka te je ažuriranje sporije za 17.71 % u odnosu na ažuriranje bez sekundarne replike. Unos podataka je sporiji za 2% nakon dodavanja sekundarne replike u sinkronom načinu rada, dok dodavanje asinkrone replike nema nikakvog utjecaja na performanse prilikom unosa podataka. Dodavanje sekundarne replike, bilo u sinkronom ili asinkronom načinu rada, nema utjecaja na performanse prilikom čitanja podataka sa primarne replike te su minimalne razlike dobivene mjerenjima posljedica trenutnog zauzeća sustava. U svrhu testiranja i analize realnog opterećenja sustava izrađena je *web* aplikacija koja predstavlja *web* blagajnu, a koristi baze podataka ovog sustava. Aplikacija je izrađena koristeći *ASP.NET MVC* i *Entity Framework* tehnologije. Provedena je i analiza utjecaja dodavanja sekundarne replike za čitanje. *AlwaysOn* konfiguracija s replikom za čitanje znatno rasterećuje sistemske resurse primarne replike (procesor, disk, memorija) te tako oslobađa resurse primarne replike za poslovno kritične operacije. Izvršavanje upita na replici za čitanje se pokazalo znatno bržim u odnosu na izvršavanje istog upita na primarnom poslužitelju za vrijeme uobičajenog produkcijskog opterećenja. Tako je izvršavanje *read-only* upita na sekundarnoj replici za čitanje brže u prosjeku 360 % u odnosu na izvršavanje istog upita na primarnoj replici. Ovdje se primijeti velika razlika u trajanju izvršavanja upita jer na sekundarnoj replici nema blokiranja i zaključavanja tablica drugim upitima kao što je to slučaj na primarnoj replici. Kako bi se utvrdila pouzdanost i dostupnost ovakvog sustava te razina dostupnosti, provedena su mjerenja potrebna za izračun RTO i RPO vrijednosti. Izračunati RTO ovog sustava iznosi 15 sekundi dok RPO iznosi svega 1 sekundu pri sinkronom načinu rada, odnosno 2 sekunde pri asinkronom načinu rada, što potvrđuje pretpostavku da je *AlwaysOn* jedan od najpouzdanijih sustava visoke dostupnosti.

Popis kratica

AG	Availability Group	grupe visoke dostupnosti
AOAG	AlwaysOn Availability Groups	AlwaysOn grupe visoke dostupnosti
AG LISTENER	Availability Group Listener	slušaoc dostupnih grupa
CRUD	Create, read, update and delete	kreiranje, čitanje, ažuriranje i brisanje
DR	Disaster Recovery	oporavak od katastrofe
HA	High Availability	visoka dostupnost
MVC	Model-View-Controller	Model-Pogled-Upravitelj
ORM	Object-relational mapping	objektno-relacijsko mapiranje
RDBMS	Relational database management system	sustav za upravljanje bazama podatka
RLO	Recovery Level Objective	dozvoljena razina gubitka
RPO	Recovery Point Objective	dozvoljeni gubitak podataka
RTP	Recovery Time Objective	dozvoljena nedostupnost sustava
SAN	Storage Area Network	umrežena spremišta podatka
SLA	Service-Level Agreements	ugovor o razini dostupnosti
VNN	Virtual Network Name	virtualno mrežno ime
WSFC	Windows Server Failover Cluster	

Popis slika

Popis tablica

Popis kôdova

Literatura

[1] UTTAM PARUI, VIVEK SANIL, Pro SQL Server Always On Availability Groups, Apress, 2016.

[2] P. A. CARTER, SQL Server AlwaysOn Revealed, 2nd Edition, Apress, 2016

[3] V. ISAKOV, Administering a SQL Database Infrastructure, Pearson Education,Inc, 2018.

[4] DMITRI KOROTKEVITCH, Pro SQL Server Internals, 2nd Edition, Apress, 2016

[5] MICROSOFT SQL DOCS, Overview of AlwaysOn Availability Groups (SQL Server), https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/overview-of-always-on-availability-groups-sql-server?view=sql-server-2017, 5.12.2019.

[6] MICROSOFT SQL DOCS, Business continuity and database recovery - SQL Server; (2017); https://docs.microsoft.com/en-us/sql/database-engine/sql-server-business-continuity-dr?view=sql-server-2017, 6.10.2019.

[7] MICROSOFT SQL DOCS, Windows Server Failover Clustering with SQL Server, https://docs.microsoft.com/en-us/sql/sql-server/failover-clusters/windows/windows-server-failover-clustering-wsfc-with-sql-server?view=sql-server-2017, 5.12.2019.

[8] PRIYANKA DEVRE, Configuring a SQL Server AlwaysOn High Availability Group, https://www.sqlshack.com/configuring-a-sql-server-alwayson-high-availability-group/, 21.11.2019.

[9] G. DANIEL, Prerequisites for Installing SQL Server AlwaysOn, https://www.sqlrx.com/prerequisites-for-installing-sql-server-alwayson/, 11.12.2019.

[10] A. CHRISTIAN, Configure Always on AG with SQL 2017, https://www.sqlservercentral.com/blogs/configure-always-on-ag-with-sql-2017, 21.11.2019.

[11] MICROSOFT SQL DOCS, Database Mirroring (SQL Server), https://docs.microsoft.com/en-us/sql/database-engine/database-mirroring/database-mirroring-sql-server?view=sql-server-2017, 9.10.2019.

[12] MICROSOFT SQL DOCS, About Log Shipping (SQL Server), https://docs.microsoft.com/en-us/sql/database-engine/log-shipping/about-log-shipping-sql-server?view=sql-server-2017, 7.10.2019.

[13] MICROSOFT SQL DOCS, Configure read only access to a secondary replica of an Always On availablility group, https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/configure-read-only-access-on-an-availability-replica-sql-server?view=sql-server-2017, 16.12.2019.

[14] MICROSOFT SQL DOC, Monitor performance for Always On availability groups, https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/monitor-performance-for-always-on-availability-groups?view=sql-server-2017, 23.1.2020.

[15] MICROSOFT SQL DOCS, Implement CRUD Functionality with the Entity Framework in ASP.NET, https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-

started/getting-started-with-ef-using-mvc/implementing-basic-crud-functionality-with-the-entity-framework-in-asp-net-mvc-application, 20.12.2019.

[16] IMPERVA, Disaster recovery solutions, https://www.imperva.com/learn/availability/disaster-recovery/, 6.10.2019.

[17] W3RESOURCE, MVC Architecture, https://www.w3resource.com/asp.net/mvc-architecture.php, 15.1.2020.

[18] TUTORIALSTEACHER, MVC, https://www.tutorialsteacher.com/mvc/mvc-controller; 15.1.2020.

[19] STACKEXCHANGE, How to query Mirroring RTO & RPO from system objects, https://dba.stackexchange.com/questions/190467/how-to-query-mirroring-rto-rpo-from-system-objects, 20.1.2020.

[20] MICROSOFT SQL DOCS, Monitor Performance for AlwaysOn Availability Groups https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2012/dn135338(v=sql.110)?redirectedfrom=MSDN, 20.1.2020.

[21] SQLSHACK, Measuring Availability Group synchronization lag, https://www.sqlshack.com/measuring-availability-group-synchronization-lag/, 20.1.2020.

[22] SQL SERVER TECHNICAL ARTICLE, AlwaysOn: Offloading Read-Only Workloads to Secondary Replicas;22.1.2020.

Prilog

Kao prilog ovog završnog rada priložen je završni rad u .docx i .pdf formatu, upute za instalaciju i konfiguraciju *AlwaysOn* sustava te web aplikaciju izrađenu za demonstraciju rada s *AlwaysOn* sustavom visoke dostupnosti. Svi prilozi nalaze se na DVD-u priloženom ovom radu.

ALGEBRA
VISOKO UČILIŠTE

OSIGURAVANJE VISOKE DOSTUPNOSTI BAZA PODATAKA KORISTEĆI MICROSOFT TEHNOLOGIJE

Pristupnik: Vladimir Gladoić, 0661047197

Mentor: mr. sc. Mario Fabijanić