Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.4
DOI: https://doi.org/10.24867/28BE07Malesevic

SISTEM ZA PREGLED I PRETRAGU VESTI BAZIRAN NA GRAPHQL API-U I RAVENDB-U

SYSTEM FOR VIEWING AND SEARCHING NEWS BASED ON GRAPHQL API AND RAVENDB

Igor Malešević, *Fakultet tehničkih nauka, Novi Sad*

Oblast – PRIMENJENE RAČUNARSKE NAUKE I INFORMATIKA

Kratak sadržaj – *Ovaj rad prikazuje pristup razvoju aplikacije za pretraživanje vesti koja se zasniva na GraphQL API-u i RavenDB-u*

Ključne reči: *GraphQL, RavenDB, Lucene Search Engine*

Abstract – *This paper proposes an approach in developing an application for viewing and searching news based on GraphQL API and RavenDB.*

Keywords: *GraphQL, RavenDB, Lucene Search Engine*

1. UVOD

U današnjem društvu, pristup informacijama je postao neizostavan deo svakodnevnog života. Razvoj interneta je doprineo ogromnoj količini podataka i znanja koji su nam lako dostupni na samo par klikova, gde god se nalazili. Zahtevi za efikasne metode pretraživanja informacija su postale normativ, kako zbog uštede vremena tako i zbog potrebe za što relevantnijim i tačnijim rezultatima. Jedna od sfera zahvaćena ovim promenama jesu mediji. Novine, kao štampani mediji, televizija i radio postepeno gube korisnike, s obzirom da su sve informacije dostupne na internetu. Iz tog razloga, većina medija su otvorili svoje onlajn veb stranice gde pružaju pregled i pretragu vesti. Na ovakvim stranicama, korisnici očekuju da mogu lako i intuitivno da dođu do informacija koje ih zanimaju uz efikasnu pretragu i filtriranje kako bi suzili skup rezultata na onaj koji je njima najpotrebniji. Još jedan od bitnih aspekata digitalizacije medija jeste proces praćenja i analiza medija, koji je bitan za razumevanje aktuelnih događaja širom sveta.

Ovakvih sistema i veb stranica ima puno, jer veliki broj medija u ovom momentu imaju svoje veb stranice i onlajn portale. Neke od popularnih globalnih veb stranica su *Google News*, *Bing News* i *News360*. Pored globalnih veb stranica, svaka zemlja ima medije koji imaju svoje veb stranice, a primeri popularnih stranih jesu *BBC News* i *CNN*, dok domaćih *RTS*, *B92* itd.

U ovom radu, rešavan je problem implementacije jedne veb stranice koja korisnicima omogućava pregled svih najnovijih vesti. Adresiran je problem velike količine podataka koji mogu dovesti do potencijalnih problema u performansama. Prilikom izrade rada, korišćen je *GraphQL* kao programski interfejs aplikacije i *RavenDB* kao baza podataka. Odabir navedenih tehnologija je odgovor na prethodno pomenute potencijalne probleme u performansama.

GraphQL je u prethodnim godinama postao dosta popularniji i zastupljeniji način korišćenja programskog interfejsa aplikacije, te su velike organizacije počele da koriste ovu tehnologiju. *Facebook*, radeći na svojim aplikacijama, susreo se sa problemom performansi kako su kompleksnost same aplikacije i broj korisnika rasli. Podaci koji se koriste na početnoj strani aplikacije i upiti ka serveru su bili veoma različiti i bilo je neophodno dobaviti podatke iz više poziva ka serveru. Ovo je inspirisalo njihove zaposlene da kreiraju *GraphQL* kao upitni jezik koji im omogućava dobavljanje podataka iz perspektive klijentske aplikacije. Ova tehnologija je zatim javno objavljena 2015. godine [1].

2. PROGRAMSKI INTERFEJS APLIKACIJE

GraphQL [2] je moderan način za kreiranje i pisanje upita za programski interfejs aplikacije *(API – Application Programming Interface).* Poseduje dvostruku ulogu - sa klijentske strane, to je upitni jezik, dok sa serverske strane predstavlja izvršni okvir *(runtime).* Klijentska aplikacija koja implementira *GraphQL* može da definiše i šalje upite na server koji mora da implementira *GraphQL* servis koji je sposoban da razume upite i odgovori na bilo koji zahtev napisan u *GraphQL* jeziku. *GraphQL* karakteristike su:

- Strogo definisani tipovi podataka;
- Tačno definisani podaci;
- Nezavisnost od verzije;
- Hijerarhija;
- Introspekcija.

2.1. Upiti

GraphQL upiti se koriste kada želimo da dobavimo podatke sa servera. Predstavljaju operaciju koja služi isključivo za čitanje podataka, što znači da ne menjaju stanje podataka u bazi. *GraphQL* upite možemo uporediti sa *GET* zahtevom u *REST API*-u. *GraphQL* upiti nam mogu pomoći u rešavanju takozvanih problema prekomernog *(over-fetching)* ili nedovoljnog dobavljanja *(under-fetching)* podataka sa servera.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Dragan Ivanović, red. prof.

2.2. Mutacije

GraphQL mutacije predstavljaju mehanizam za izvršavanje promena nad podacima na serverskoj strani. Mutacije omogućavaju klijentima da šalju promenljive ili argumente ka serveru kako bi ažurirali podatke, kreirali nove entitete ili obrisali postojeće. Za razliku od upita, mutacije su destruktivne i menjaju stanje podataka u bazi. *GraphQL* mutacije možemo uporediti sa *POST, PUT* i *DELETE* zahtevima u *REST API*-u.

2.3. Pretplate

GraphQL pretplate predstavljaju mehanizam koji omogućava komunikaciju u realnom vremenu između servera i klijenta. Pretplate omogućavaju klijentima da se pretplate na određene događaje ili promene na serveru, a zatim dobiju automatski ažurirane podatke kako se ti događaji dese. To omogućava izgradnju aplikacija kao što su četovi, obaveštenja, novosti itd, gde je komunikacija u realnom vremenu od ključnog značaja. *GraphQL* pretplate koriste *WebSocket* protokol za održavanje dugotrajnih veza između klijenta i servera.

3. SKLADIŠTENJE PODATAKA

Skladištenje podataka je suštinska funkcija baza podataka, gde informacije mogu biti organizovane u tabelama, dokumentima, grafikonima itd. u zavisnosti od vrste baze podataka. Baza podataka je centralizovano mesto gde se čuvaju podaci potrebni za funkcionisanje aplikacije.

3.1. *RavenDB*

RavenDB je dokument orijentisana *(document oriented)* nerelaciona *(NoSQL)* baza podataka koja se ističe svojom sposobnošću efikasnog rukovanja nestrukturiranim podacima. *RavenDB* čuva podatke u formatu *JSON (JavaScript Object Notation)* dokumenata, što olakšava rad sa složenim podacima i omogućuje jednostavnu integraciju sa modernim aplikacijama. Ono što karakteriše i izdvaja ovu nerelacionu bazu jesu *ACID* osobine: [3]

- Atomičnost *(Atomicity)* – sve naredbe jedne transakcije se izvršavaju u celosti ili nikako, čime se osigurava integritet podataka u bazi u slučaju dešavanja greške.
- Konzistentnost *(Consistency)* – baza podataka mora biti u konzistentnom stanju i pre i posle izvršavanja transakcije. Jedna transakcija dovodi bazu podataka iz jednog u drugo konzistentno stanje.
- Izolacija *(Isolation)* – svaka transakcija je izolovana i zaštićena od istovremeno izvođenih drugih konkurentnih transakcija nad istim podacima.
- Trajnost *(Durability)* – Garancija da će svaka uspešno izvršena transakcija biti zapamćena u bazi podataka i u slučaju sistemske greške.

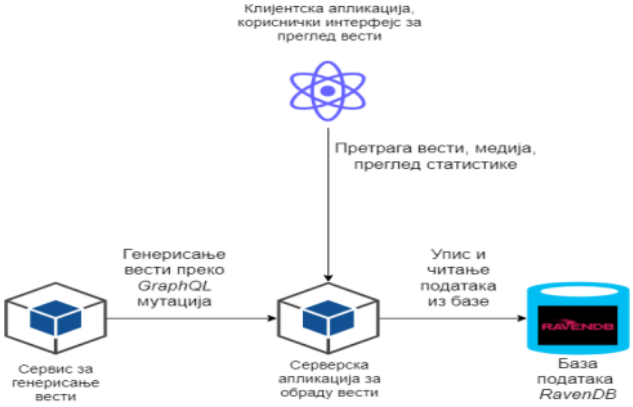
RavenDB koristi *Lucene* biblioteku za pronalaženje informacija *(Lucene Search Engine)*. Ova biblioteka predstavlja moćan i otvoren izvor pretraživačkih tehnologija koje omogućavaju brzu i efikasnu pretragu tekstualnih podataka. Indeksiranje predstavlja proces organizovanja i skladištenja podataka u strukturu koja omogućava brzu i efikasnu pretragu. Osnovna ideja indeksiranja je transformacija tekstualnih podataka u optimizovan format koji omogućava brzu pretragu i pruža informacije o lokacijama gde se ti podaci nalaze.

RavenDB radi sa indeksima koji predstavljaju serverske funkcije koje definišu nad kojim poljima i vrednostima iz dokumenta je moguće raditi pretragu. Indeksi predstavljaju jedini način da se izvrše upiti u *RavenDB*-u. Oni se mogu podeliti naspram izvora na statičke i dinamičke, dok se naspram funkcionalnosti dele u tri grupe: [4]

1. *Map Index* – sadrži jednu funkciju mapiranja koja ukazuje koja polja iz dokumenta treba da budu indeksirana.
2. *Multi Map Index* – sadrži više funkcija mapiranja koje dozvoljavaju da se indeksiraju podaci iz više različitih kolekcija (npr. polimorfističnih).
3. *Map Reduce Index* – indeksi koji omogućavaju kompleksne agregacije podataka uz brzo izvršavanje upita nad njima. Pri dodavanju ili izmeni podataka, ovaj indeks će odraditi ponovno kalkulisanje i uvek imati najsvežije podatke. Indeks se sastoji iz dve faze:
    a. Faza mapiranja – izvršava funkcije mapiranja nad dokumentima iz navedenih kolekcija.
    b. Faza stapanja – izvršava funkciju grupisanja podataka po određenom kriterijumu, a zatim radi agregaciju podataka.

4. SPECIFIKACIJA SISTEMA

U ovom poglavlju je predstavljena arhitektura sistema prikazana na slici 4.1.



Slika 4.1. *Arhitektura sistema*

U sistemu se može identifikovati nekoliko komonenti:

1. Servis za generisanje vesti – Ova komponenta predstavlja serversku aplikaciju koja generiše vesti i pomoću *GraphQL* mutacija poziva serversku aplikaciju za obradu vesti. Ovaj servis je napravljen kako bi simulirao pristizanje velikog broja vesti u sistem.
2. Serverska aplikacija za obradu vesti – Ova komponenta predstavlja glavni servis ovog rada. To je serverska aplikacija koja nam nudi *GraphQL* server sa propisanom šemom i

tipovima podataka. Preko upita nudi mogućnost čitanja podataka iz baze podataka, dok preko mutacija omogućava upisivanje novih podataka. Ima konekciju ka bazi podataka, a predstavlja programski interfejs aplikacije koji konzumira klijentska aplikacija.

3. Baza podataka *(RavenDB)* – Komponenta koja nam služi za skladištenje podataka.
4. Klijentska aplikacija – Ova komponenta predstavlja korisnički interfejs preko kojeg krajnji korisnici imaju mogućnost pregleda i pretrage vesti i medija, kao i prikaz statistike.

5. IMPLEMENTACIJA

U ovom poglavlju je dat detaljniji opis implementacije svake komponente sistema.

5.1. Servis za generisanje vesti

Ovaj servis je napravljen kako bi bilo simulirano kreiranje i pristizanje vesti na serversku aplikaciju. Napravljena je kao *.NET* aplikacija koja ima svoj pozadinski zadatak *(Background job)* i okida se periodično. Period okidanja je podesiv kao i broj podataka koji se generiše. Kako bi sistem ostao u konzistentnom stanju, generišu se svi entiteti koji su potrebni za uspešno indeksiranje podataka i redosled generisanja je bitan. Unutar serverskih aplikacija je iskorišćen *CQRS (Command Query Responsibility Seggregation)* šablon [5]. Iz pozadinskog zadatka pozivamo komandu za generisanje vesti.

5.2. Serverska aplikacija za obradu vesti

Ovaj servis predstavlja glavnu serversku aplikaciju koja pruža sve potrebne funkcionalnosti, predstavlja *GraphQL* server i radi indeksiranje podataka koji se skladište u *RavenDB* bazu podataka. Konfiguracija *GraphQL* servera se radi pomoću biblioteke *HotChocolate.* Prilikom konfiguracije same *.NET* aplikacije je potrebno samo navesti da se ona koristi kao *GraphQL* server. Dodatno, pored definisanja *GraphQL* servera, potrebno je definisati tipove upita i mutacija. Ovi tipovi definišu sve moguće upite i mutacije koje je moguće odraditi nad servisom. Definisani upiti i mutacije se oslanjaju na definisane tipove podataka, ulazne argumente i povratne vrednosti. Sve ovo čini *GraphQL* šemu [6].

Validacija podataka i upravljanje greškama je bitan proces unutar ovog servisa. Ovaj proces predstavlja osiguranje da su uneseni podaci u saglasnosti sa zadatim standardima, formatima i pravilima, što sprečava unos pogrešnih ili nevalidnih podataka u sistem i dovođenje sistema u nekonzistentno stanje. Zbog svojih karakteristika, *GraphQL* nudi validaciju na više nivoa, a to obuhvata:

- Semantičku validaciju upita;
- Validaciju tipova podataka;
- Validaciju ulaznih argumenata;

Sledeći nivo validacije podataka predstavlja ispravnost podataka naspram domenske i biznis logike. Ukoliko ova ispravnost podataka nije ispoštovana, takođe je potrebno sprečiti upit da se izvrši i pozivaoca upita je potrebno obavestiti o grešci.

5.3. *RavenDB* baza podataka

Pokretanjem skripte, moguće je dobiti pristup bazi podataka u lokalnom okruženju. Za konfiguraciju konekcije sa bazom podataka, prvo je neophodno dodati zavisnost *(dependency)* na biblioteku *RavenDB.Client.* Dodavanjem ove biblioteke dobija se pristup klasama i metodama koje omogućavaju laku konfiguraciju konekcije. Za uspešnu konekciju, neophodna su dva podatka - *URL* do servera baze i ime konkretne baze podataka nad kojom se izvršavaju upiti.

5.4. Klijentska aplikacija

Ovaj servis predstavlja korisnički interfejs koji konzumira podatke sa serverske aplikacije za obradu vesti i prikazuje te podatke korisnicima. Klijentska aplikacija je implementirana u *React* radnom okviru. Za stilizaciju i komponente interfejsa korišćena je biblioteka *Fluent UI React Northstar*. Dodatno, korišćene su biblioteke koje omogućavaju upravljanje stanjem podataka, tokom podataka i *GraphQL*-om, a one su:

- *React Redux Toolkit* – biblioteka koja predstavlja skup alata koji omogućavaju implementaciju upravljanja stanjem unutar *JavaScript* aplikacija, olakšavajući pristup podacima i ažuriranje podataka. Prednost ove biblioteke dolazi do izražaja u aplikacijama koje koriste veliki broj komponenti i istovremeno moraju da ažuriraju korisnički interfejs na više delova [7].
- *Redux Saga* – biblioteka za upravljanje asinhronim radnjama u *Redux*-u kao što su *HTTP* zahtevi, promene stanja i obrada bočnih efekata.
- *GraphQL Apollo Client* – biblioteka koja se koristi u razvoju veb aplikacija zasnovanih na *GraphQL*-u. Ova biblioteka omogućava laku integraciju i upravljanje *GraphQL*-om zahvaljujući svojim karakteristikama:
    - Lako izvršavanje upita i mutacija.
    - Keširanje što omogućava brze upite i sprečava ponovne zahteve ka serveru za iste podatke.
    - Upravljanje stanjem aplikacije.
    - Integracija sa modernim radnim okvirima.
- *Codegen* – biblioteka koja omogućava korišćenje karakteristike *GraphQL*-a introspekcije, tj. Omogućava da se na klijentskoj strani automatski izgenerišu šema, upiti, mutacije i tipovi podataka koje server nudi. Biblioteka uzima u obzir *GraphQL* fajlove kreirane na klijentskoj strani. Zahvaljujući njoj, eliminiše se potreba za ručnim pisanjem svih navedenih stavki, čime se programeru ubrazava rad sa *GraphQL*-om.

6. ZAKLJUČAK

U ovom radu se mogao videti način implementiranja jedne veb aplikacije za prikaz vesti, koje su postale svakodnevnica procesom digitalizacije. Ono na čemu je bio akcenat jesu performanse takve aplikacije. Rešenje takvog problema je adresirano na dva načina, a to su

korišćenje *GraphQL API*-a kao korisničkog interfejsa aplikacije i korišćenje baze podataka koja se oslanja na *Lucene Search Engine* kako bi podržala upite i pretrage nad punim tekstom.

GraphQL kao korisnički interfejs aplikacije omogućio je strogo definisanje šeme, upita, mutacija i tipova podataka, čime klijentska aplikacija može da kreira upite i na taj način da odabere samo one podatke koji su joj zaista potrebni. Dobavljanje preterane količine *(over-fetching)* i male količine (*under-fetching)* podataka su problemi koji su rešeni korišćenjem *GraphQL*-a.

RavenDB, kao odabrana baza podataka omogućila je da se uz visoke performanse izvršavaju upiti i skladište podaci. Kako koristi *Lucene Search Engine* u pozadini, to je omogućilo indeksiranje podataja na onaj način koji je optimalan za pretragu vesti koje pristižu u sistem u velikim količinama. Pored pretrage vesti, jako bitnu ulogu je odigrala u geolokacijskoj pretrazi i statistikama.

U ovom radu nije iskorišćen *WebSocket* protokol i *GraphQL* pretplate, te se to može izdvojiti kao potencijalno unapređenje sistema. *GraphQL* pretplate omogućavaju komunikaciju servera sa klijentskom aplikacijom. Prilikom pristizanja novih vesti u sistem bi bilo moguće obavestiti klijentsku aplikaciju o promenama i na neki način korisniku staviti do znanja da postoje najnovije vesti. Korisnički interfejs bi mogao odmah da prikaže novu vest na stranici za prikaz, međutim sa strane korisničkog iskustva, ovo može da bude problem ukoliko vesti pristižu u kratkim intervalima. U tom slučaju, obaveštenje sa određenim detaljima bi bilo dovoljno da se prikaže na korisničkom interfejsu. Dodatno, kako se ne bi gomilali nepotrebni podaci u sistemu i time opterećivali bazu podataka i uticali na performanse sistema, mogli bi se implementirati pozadinski zadaci koji bi se okidali svaki dan i čistili bazu od podataka koji nisu više relevantni.

7. LITERATURA

[1] Facebook introducing GraphQL, https://engineering.fb.com/2015/09/14/core-infra/graphql-a-data-query-language/
[2] GraphQL, official documentation, https://graphql.org/
[3] RavenDB, ACID database, official documentation https://ravendb.net/why-ravendb/acid-transactions
[4] RavenDB Indexes, official documentation, https://ravendb.net/docs/article-page/5.4/csharp/indexes/what-are-indexes
[5] CQRS Pattern, official Microsoft documentation, https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs
[6] Schema defintion, https://graphql.org/learn/schema/
[7] Marc Garreau, Redux in action, 2018, ISBN 9781617294976

Kratka biografija:

Igor Malešević rođen je 19.01.1998. godine u Beogradu. Godine 2016. upisao je Fakultet Tehničkih Nauka u Novom Sadu, odsek Računarstvo i automatika. Osnovne studije završio je u septembru 2020. Od oktobra 2020. Upisuje Master akademske studije.