Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/26BE03Banjac

EFIKASNO UPRAVLJANJE PODACIMA U VEB APLIKACIJAMA UPOTREBOM *REACT QUERY* BILBLIOTEKE

EFFICIENT DATA MANAGEMENT IN WEB APPLICATIONS USING THE *REACT QUERY* LIBRARY

Damjan Banjac, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu predstavljena je biblioteka React Query. Naglasak je na njenim osnovnim principima, mogućnostima primene, prednostima i nedostacima. Razmatrani su neki od glavnih aspekata ove biblioteke međukojima je, pre svega, omogućavanje efikasnog izvršavanja HTTP zahteva za dobijanje podataka sa servera. Njegova jednostavna i intuitivna sintaksa olakšava upravljanje asinhronim operacijama, a ugrađena podrška za keširanje podataka na klijentskoj strani omogućava brzo pristupanje i ažuriranje podataka uklanjanjem nepotrebnih zahteva ka serveru.*

Ključne reči: *React Query, veb aplikacija, upravljanje podacima, stanje aplikacije*

Abstract – *This paper introduces the React Query library, focusing on its fundamental principles, practical applications, advantages, and disadvantages. It discusses some of the key aspects of this library, highlighting its capability to efficiently execute HTTP requests for fetching data from the server. Its simple and intuitive syntax makes managing asynchronous operations easier, and built-in support for client-side data caching enables quick access and updates to data without the need for unnecessary server requests.*

Keywords: *React Query, web application, data management, application state*

1. UVOD

U poslednjim decenijama veb aplikacije postaju sve važniji deo našeg svakodnevnog života. One nam pružaju različite usluge i sadržaj. S obzirom na rast kompleksnosti i količine podataka koje veb aplikacije obrađuju, performanse i efikasnost postaju ključni faktori za korisničko iskustvo. U isto vreme, upravljanje stanjem aplikacije, odnosno kako se podaci sinhronizuju između servera i klijenta, predstavlja izazov zbog potrebe za brzim i konzistentnim prikazom ažuriranih informacija.

Do sada primenjivani pristupi upravljanju stanjem, kao što su REST (*Representational State Transfer*) interfejs i biblioteke za upravljanje stanjem poput Redux biblioteke, suočavaju se sa ograničenjima i problemima u vezi sa performansama i složenošću koda.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bila dr Dunja Vrbaški, docent.

REST programski interfejsi često su dovodili do problema preuzimanja suviše mnogo ili premalo podataka (eng. *over-fetching, under-fetching*), što je rezultiralo nepotrebnim opterećenjem mreže i dužim vremenom odziva aplikacije.

Da bi se rešili ovih problema, pojavila se biblioteka React Query kao moćno i jednostavno rešenje za optimizaciju performansi i upravljanje stanjem u React aplikacijama. React Query koristi tehnike keširanja podataka, paginacije i ranijeg preuzimanja kako bi se izbeglo nepotrebno preuzimanje i obrada podataka sa servera. Ova biblioteka takođe pruža koncizan API za slanje zahteva serveru i upravljanje stanjem čime se smanjuje nepotrebni ponavljajući kod i povećava efikasnost aplikacije.

U ovom radu se istražuje i analizira verzija 3 TanStack React Query biblioteke, pri čemu se ističu njeni ključni prednosti i nedostaci u poređenju sa tradicionalnim pristupima upravljanju stanjem i performansama. Demonstrirane su praktične funkcionalnosti i primene ove biblioteke u stvarnom okruženju.

2. TEORIJSKA OSNOVA

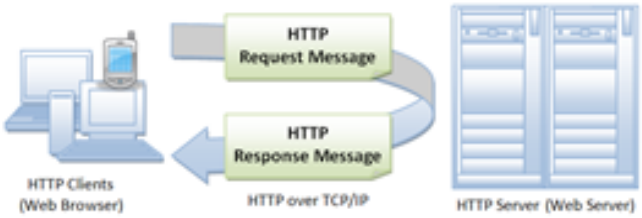
Veb aplikacija se sastoji iz dva ključna dela, takozvani: *front-end* i *back-end*. Prednji deo se odnosi na korisnički interfejs, uključujući HTML, CSS i JavaScript, koji korisnici vide i koriste za interakciju. Zadnji deo, sa druge strane, je skrivena infrastruktura koja radi iza scene. Ona se sastoji od serverskih komponenata i programa napisanih u jezicima kao što su: Python, Ruby, PHP, Java i Node.js. On obrađuje zahteve korisnika, komunicira sa bazom podataka i štiti podatke.

Stanje aplikacije, koje podrazumeva *client state* i *server state*, je ključni koncept u upravljanju podacima u veb aplikacijama. Stanje na klijentskoj strani, obuhvata podatke čuvane na uređaju korisnika, poput kolačića (engl. cookies) ili lokalnog skladišta (engl. local storage), za pamćenje preferencija i sesija. Stanje na serverskoj strani podrazumeva podatke na serveru ili u bazi podataka, uključujući informacije o korisnicima, proizvodima i drugim aplikacijskim podacima [1].

Stanje u veb aplikacijama obuhvata informacije i podatke koji se održavaju tokom interakcije korisnika s aplikacijom, ključno utičući na ponašanje i usluge aplikacije.

HTTP (*Hypertext Transfer Protocol*) je protokol za razmenu podataka između klijenata i veb servera putem

interneta. Radi na principu zahteva i odgovora gde klijent šalje zahtev za resursima, a server odgovara statusnim kodom, zaglavljima i telom odgovora. HTTP često koristi TCP/IP sloj za pouzdanu komunikaciju između klijenta i servera (Slika 1).



Slika 1. *Princip rada HTTP [11]*

3. UPOZNAVANJE SA REACT QUERY BIBLIOTEKOM

U nastavku su ukratko predstavljene funkcionalnosti biblioteke React Query.

3.1. Instalacija React Query

React Query biblioteka se može instalirati korišćenjem npm ili yarn komandi:

- npm install react-query,
- yarn add react-query [2].

3.2. Osnovni koncepti React Query biblioteke

React Query je biblioteka za upravljanje stanjem i za dobavljanjem podataka u React aplikacijama. Osnovni koncepti ove biblioteke su:

- Upit (*Query*) predstavlja zahtev za dobijanje podataka sa servera.
- Mutacije (*Mutation*) su zahtevi za promenom podataka na serveru, kao što su kreiranje, ažuriranje ili brisanje podataka [3].
- Funkcija upita (*Query Function*) je funkcija koja definiše logiku za izvršavanje upita.
- *UseQuery hook* se koristi u React komponentama kako bi se izvršio upit i dobili podaci.
- *UseMutation hook* se koristi za izvršavanje mutacija.
- Keširanje podataka je tehnika čuvanja kopije podataka u brzom i privremenom skladištu, odnosno u keš memoriji.
- Invalidacija keša se odnosi na automatsko osvežavanje podataka u keš memoriji nakon izvršavanja mutacija kako bi se odražavale promene na serveru [2].

3.3. Razmatranje ključnih funkcionalnosti

React Query pruža različite mogućnosti za izvođenje zahteva ka serveru i keširanje podataka kako bi se poboljšala efikasnost i performanse upravljanja podacima u React aplikacijama.

Najbitnije funkcionalnosti koje nudi React Query su izvođenje zahteva prema serveru, paginacija i vraćanje unapred.

Izvođenje zahteva prema serveru se može praktikovati pomoću takozvanih udica (eng. *hook*). Neke od tih udica su: *useQuery Hook, useMutation Hook i useInfiniteQuery Hook* [4].

Operacija za keširanje podataka koje podržava React Query su: automatsko keširanje, kontrola keša i invalidacija keša. React Query automatski kešira podatke preuzete putem *useQuery* i *useInfiniteQuery*. To znači da će svaki sledeći put, kada se isti upit izvrši, podaci biti brzo preuzeti iz keša umesto slanja novog zahteva serveru. Pod terminom kontrola keša se misli na različite opcije za kontrolu keš memorije i njenih podataka. Na primer, može se postaviti opcija *staleTime* čime se određuje koliko dugo podaci ostaju važeći pre nego što se ponovo preuzimaju sa servera. Takođe, može se koristiti i opcija *refetchInterval* kako bi se podaci u keš memoriji automatski osvežavali u redovnim intervalima. Invalidacija keša je proces ručnog osvežavanja podataka kako bi se odražavali satnje i preuzeli novi podaci sa servera. React Query omogućava invalidaciju keša pozivom metode *queryClient.invalidateQueries()* [4].

Paginacija, straničenje, je tehnika koja omogućava prikazivanje velikog broja podataka u sekcijama ili delovima, odnosno stranicama. U React Query biblioteci paginacija se može izvršiti pomoću *useInfiniteQuery* i *usePaginatedQuery* [5].

Preuzimanje unapred (engl. *prefetching*) je tehnika u veb aplikacijama koja omogućava preuzimanje podataka sa servera pre nego što ih korisnik eksplicitno zatraži. Ova tehnika ima za cilj poboljšanje performansi i korisničkog iskustva smanjenjem vremena čekanja na podatke tokom interakcije sa aplikacijom [6]. Iako je ovo korisna tehnika sa potencijalom za značajno poboljšanje rada i funkcionalnosti veb aplikacija, neki od potencijalnih problema koje treba razmotriti su nepotrebno opterećenje servera i mreže, povećana potrošnja resursa, zastareli podaci i složenost implementacije.

4. POREĐENJE SA DRUGIM BIBLIOTEKAMA

Kada se pričalo o upravljanju stanjem podataka u veb aplikacijama, pre pojave React Query biblioteke, najpopularnija rešenja za ovaj problem nudili su: Redux i Apollo Client.

U poređenju sa Redux bibliotekom, React Query predstavlja bolji alat za upravljanje stanjem podataka, a glavne prednosti su to što ima manje šablonskog, ponavljajućeg koda, što implicira da je jednostavniji za integraciju u projekte. Takođe, ima mogućnost automatskog keširanja za razliku od Redux biblioteke koja ne podržava tu opciju [7].

Kada je u pitanju Apollo Client, on je napravljen za rad isključivo sa GraphQL projektima, pa konkretno zbog toga React Query predstavlja multifunkcionalniju i više prilagodljivu opciju za bilo koju vrstu projekta [8].

5. OPTIMIZACIJA PERFORMANSI

Automatska optimizacija upita u React Query ima za cilj da pojednostavi i poboljša performanse rada sa asinhronim podacima, omogućavajući programerima da se fokusiraju na razvoj komponenata umesto na kompleksno upravljanje stanjem i keširanjem. Pod automatske optimizacije upita u React Query se podrazumevaju: automatsko keširanje i osvežavanje, poništavanje keša nakon mutacija, optimističko ažuriranje i optimizacija preuzimanje novih strana [9].

Takođe, React Query nudi mogućnost ručnog podešavanja keširanja za specifične upite ili mutacije ako to zahteva aplikacija. Keš memorija se može ručno podesiti pomoću metoda: *Invalidate, Refech i Manual Query Creation*.

Metoda *Invalidate* je moćna metoda koja omogućava ručno poništavanje keširanih podataka za određeni upit [10]. Metoda *Refech* omogućava ručno ponovno izvršavanje upita i osvežavanje keširanih podataka. Metoda *Manual Query Creation* omogućava precizno ručno podešavanje opcija keš memorije.

6. REAKTIVNOST I PRILAGODLJIVOST

Praćenje stanja upita i promena u React Query biblioteci je ključno za razumevanje trenutnog statusa izvršavanja upita i reagovanje na promene u trenutnom stanju. Stanja se prate na nekoliko načina korišćenjem različitih komandi:

- Korišćenjem: *isLoading, isError, isSuccess* [4];
- Korišćenjem: *onSuccess, onError i onSettled* opcija;
- Upotrebom: *useQueryClient* za praćenje promena
- Komandom: *useIsFetching* za praćenje aktivnih upita

Rad sa podacima u realnom vremenu uz React Query biblioteku omogućava brzo osvežavanje podataka. Koristi se za prikazivanje ažuriranih podataka bez ručnog osvežavanja stranice. React Query podržava ovo kroz invalidaciju keša i procesima udruživanja (eng. *polling*).

Korišćenje parametara za dinamičke upite u React Query biblioteci predstavlja koristan mehanizam za prilagodljivo i fleksibilno preuzimanje podataka sa servera ili drugih izvora. Umesto statičkog definisanja upita sa fiksnim vrednostima, ovaj pristup omogućava generisanje upita na osnovu dinamičkih parametara čime se omogućava višestruka upotreba upita u različitim situacijama.

Dodatno, upiti koji su parametrizovani olakšavaju održavanje koda jer, umesto kreiranja više upita za svaku moguću varijaciju, može se koristiti isti upit uz promenu vrednosti parametara. To omogućava ponovno iskoristive i prilagodljive upite, smanjuje dupliranje koda i olakšava prilagođavanje promenama u zahtevima.

Upravljanjem sa više paralelnih upita u React Query biblioteci omogućava efikasno koordiniranje i izvršavanje više nezavisnih upita ka serveru ili drugim izvorima podataka istovremeno čime se postiže brži prikaz informacija i bolje performanse aplikacije. Ova funkcionalnost je od velikog značaja u situacijama gde su podaci međusobno nezavisni i mogu se preuzimati nezavisno jedan od drugog čime se smanjuje ukupno vreme čekanja.

7. INTEGRACIJA SA DRUGIM TEHNOLOGIJAMA

Integracija React Query biblioteke sa React bibliotekom i komponentama omogućava efikasno upravljanje stanjem i preuzimanjem podataka unutar React aplikacija. Ova integracija omogućava: prilagođavanje performansi, unapređenje korisničkog iskustva i jednostavnij dalji razvoj.

Korišćenje React Query biblioteke sa React Router komponentom omogućava dalju, bolju implementaciju i upravljanje stanjem kao i preuzimanje podataka unutar aplikacije koja koristi rutiranje. Integracija ovih dveju biblioteka pruža mogućnost dinamičkog upravljanja podacima na osnovu trenutne rute omogućavajući još precizniji prikaz i ažurnije informacije. Kroz upotrebu *useQueryClient* i *useNavigate* hukova, koje pruža React Router, mogu se efikasno povezati akcije korisnika sa upitima i promenama u keširanim podacima.

Autentikacija i autorizacija uz pomoć React Query biblioteke pružaju koristan način za upravljanje sigurnošću i pristupom u React aplikacijama. React Query omogućava efikasnu implementaciju mehanizama autentikacije i autorizacije kako bi se zaštitili resursi i omogućio pristup određenim elementima aplikacije samo ovlašćenim korisnicima .

8. PRIMENA REACT QUERY U REALNIM APLIKACIJAMA

U nastavku su navedeni primeri tipova aplikacija u kojima se mogu primeniti funkcionalnosti React Query biblioteke čime se mogu poboljšati njihove performanse.

U aplikacijama za društveno umrežavanje se React Query može primeniti za:

- prikaz informacija na vremenskoj liniji,
- upravljanje profilima korisnika,
- obaveštenja u realnom vremenu,
- pretragu i filtriranje sadržaja
- beskonačno listanje
- optimizaciju performansi.

U sličnom kontekstu, kao i kod aplikacija za društveno umrežavanje, React Query nudi i podršku za aplikacije iz oblasti e-trgovine, i to za sledeće aktivnosti:

- dinamičko preuzimanje podataka o proizvodima,
- pretragu i filtriranje proizvoda ili usluga,
- straničenje za pregled kataloga,
- kreiranje korpe i liste želja,
- personalizovane predloge proizvoda.

Veoma velik potencijal za primenu React Query ima i kod aplikacija za rezervaciju putovanja jer kroz kombinaciju brzog preuzimanja i keširanja informacija o destinacijama, smeštaju, transportu i cenama, React Query omogućava aplikacijama da pruže korisnicima trenutan i tačan pregled dostupnih opcija za putovanja.

9. ZAKLJUČAK

React Query omogućava efikasno izvršavanje zahteva za preuzimanje i ažuriranje podataka sa servera. Ugrađena podrška za keširanje podataka na klijentskoj strani omogućava brz pristup i smanjenjne broja i veličine nepotrebnih zahteva prema serveru. Time se smanjuje opterećenje na serveru i poboljšava se brzina odziva aplikacije. Karakteristike React Query bibliotee, istražene i testirane kroz ovaj rad, ukazuju na to da React Query može biti ključni alat za razvoj modernih veb aplikacija.

10. LITERATURA

[1] Florin Software Consulting, „Server State vs Client State“, https://www.florinsoftwareconsulting.com/blog/server-vs-client-state (pristupljeno u avgustu 2023.)

[2] Edidiong Etok, „Core Concepts of React Query“, https://blog.devgenius.io/core-concepts-of-react-query-567c91035331 (pristupljeno u avgustu 2023.)

[3] Tanstack Query official documentation, „Mutation“, https://github.com/TanStack/query/blob/main/docs/react/guides/mutations.md (pristupljeno u avgustu 2023.)

[4] Tanstack Query official documentation https://tanstack.com/query/v3/docs/react/guides/ (pristupljeno u avgustu 2023.)

[5] Blitz, „usePaginatedQuery“, https://blitzjs.com/docs/use-paginated-query (pristupljeno u avgustu 2023.)

[8] React Query Firebase, „Prefetching“, https://react-query-firebase.invertase.dev/firestore/prefetching, (pristupljeno u avgustu 2023.)

[9] Tien Nguyen, „React Query vs Redux: A Detailed Comparison for Developers“; https://www.frontendmag.com/insights/react-query-vs-redux-comparison/ (pristupljeno u septembru 2023.)

[10] Tien Nguyen, „React Query vs Apollo Client: Which One Should You Use?“, https://www.frontendmag.com/insights/react-query-vs-apollo-client/ (pristupljeno u septembru 2023.)

[11] HyperText Transfer Protocol https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/http_basics.html

Kratka biografija:

Damjan Banjac rođen je 14. novembra 1997. godine u Novom Sadu. Godine 2016. upisao je Fakultet tehničkih nauka, odsek Računarstvo i automatika. Oktobra 2020. godine je diplomirao. Iste godine upisao je master studije na Fakultetu tehničkih nauka u Novom Sadu, odsek Računarstvo i automatika, studijski program Elektronsko poslovanje. Master rad odbranio je 2023. godine.