UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET

Mobilna aplikacija za anketiranje – Progressive Web aplikacija korišćenjem React i .NET tehnologija

Završni rad
Studijski program:
Elektrotehnika i računarstvo
Modul: Računarstvo i informatika

Student:

Nenad Kragović, br. ind. 14711

Mentor:

doc. dr Miloš Bogdanović

Niš, mart 2023. godina

Univerzitet u Nišu
Elektronski Fakultet

Mobilna aplikacija za anketiranje – Progressive Web aplikacija korišćenjem React i .NET tehnologija

Mobile polling application – Progressive Web application using React and .NET technologies

Završni rad
Studijski program:
Elektrotehnika i računarstvo
Modul: Računarstvo i
informatika

Student: Nenad Kragović, br. ind.14711

Mentor: doc. dr Miloš Bogdanović

Zadatak: *Istražiti arhitekturu i tehnologije za razvoj Progressive Web Aplikacija (PWA). Istražiti mogućnosti korišćenja push notifikacija u okviru Progressive Web Aplikacija na platformama poput Windows-a, Android-a i iOS-a. U praktičnom delu, korišćenjem React i .NET tehnologija, implementirati Progressive Web Aplikaciju za anketiranje korisnika sa mogućnošću definisanja različitih anketa sastavljenih od predefinisanih tipova pitanja proizvoljne sadržine.*

Datum prijave rada: _____

Datum predaje rada: _____

Datum odbrane rada: _____

Komisija za ocenu i odbranu:

_____

1. Predsednik Komisije

_____

2. Član

_____

3. Član

Mobilna aplikacija za anketiranje – Progressive Web aplikacija korišćenjem React i .NET tehnologija

SAŽETAK

Cilj ovog rada je kreiranje kompletnog rešenja za push obaveštenja kod Progressive Web Aplikacija. Progressive Web aplikacije su Web bazirane mobilne aplikacije koje se mogu instalirati poput mobilnih i desktop aplikacija na platformama poput Windows-a, Android-a i iOS-a. Rad uključuje izradu svih potrebnih aplikacija za rešenje i demonstraciju na primeru aplikacije za anketiranje. Rešenje se oslanja na korišćenje Service Worker-a i Web Push protokola.

Mobile polling application – Progressive Web application using React and .NET technologies

ABSTRACT

The goal of this work is to create a complete solution for push notifications for Progressive Web Applications. Progressive Web Applications (PWA) are Web-based mobile apps that can be installed like mobile and desktop apps on platforms like Windows, Android, and iOS. The work includes the creation of all the necessary applications for the solution and a demonstration on the example of the survey application. The solution relies on the use of Service Worker and Web Push protocol.

SADRŽAJ

1. UVOD

World Wide Web (takođe poznat kao „Web“) je globalni informacioni sistem dostupan preko interneta. Sastoji se od ogromne mreže međusobno povezanih dokumenata i drugih resursa, povezanih hiperlinkovima i URL adresama. Web omogućava korisnicima pristupanje i deljenje informacija, komunikaciju, i obavljanje različitih aktivnosti na mreži korišćenjem Web pretraživača. Web je revolucija u načinu na koji pristupamo informacijama i postao je suštinski deo modernog društva.

Web aplikacije se obično sastoje od dva glavna dela: klijentske strane (poznate i kao front-end) i serverske (takođe poznate kao back-end). Klijentska strana je odgovorna za predstavljanje informacija korisniku i rukovanje interakcijama korisnika. Obično se implementira pomoću HTML-a, CSS-a i JavaScript-a, a pokreće ga korisnikov Web pretraživač. Serverska strana je odgovorna za čuvanje i preuzimanje podataka, obradu zahteva sa klijentske strane i generisanje dinamičkog sadržaja. Klijent i server komuniciraju putem standardizovanih protokola, kao što su HTTP i HTTPS, i razmenjuju podatke u obliku zahteva i odgovora. Arhitektura na strani servera se često zasniva na obrascu Model-View-Controller (MVC), koji razdvaja logiku aplikacije na tri glavne komponente: model (predstavlja podatke i biznis logiku), view (rukuje prezentacijom podataka) i kontroler (prima korisnički unos i ažurira model i prikaz u skladu sa tim).

Pored ova dva glavna dela, Web aplikacije često koriste baze podataka za skladištenje i preuzimanje podataka, kao i razne API-je i servise za pružanje dodatnih funkcionalnosti. Arhitektura i struktura Web aplikacije mogu u velikoj meri uticati na njene performanse, skalabilnost i mogućnost održavanja, tako da su pažljivo planiranje i dizajn od suštinskog značaja.

Kako se razvijao domen Web aplikacija i klijentski hardver postajao je sve performantniji, tako se sve više logike Web aplikacije selilo sa serverskog na klijentski deo. Pojavom XMLHTTPRequest-a 2000. godine Web aplikacije su dobile mogućnost pribavljanja podataka sa URL-a bez potrebe za osvežavanjem stranice. Pet godina kasnije pojavom AJAX-a Web aplikacije su mogle da šalju i pribavljaju podatke od servera asinhrono (u pozadini) bet uticaja na prikaz i ponašanje stranice. Ovo je bio veliki korak napred za Web aplikacije, ali su one i dalje bile ograničene po pitanju offline sposobnosti, korišćenju push obaveštenja i drugih funkcionalnosti nalik native aplikacijama.

Progressive Web Aplikacije su se pojavile kao odgovor na ova ograničenja, nudeći iskustvo nalik native aplikacijama unutar Web aplikacija. PWA su Web aplikacije koje su napravljene korišćenjem Web tehnologija, ali se mogu instalirati na korisnikov uređaj baš kao native aplikacije, pružajući pristup van mreže, push obaveštenja i druge funkcije. Cilj PWA je da premosti jaz između Web i native aplikacija, pružajući korisnicima i programerima najbolje iz oba sveta.

Cilj ovog rada je kreiranje kompletnog rešenja za push obaveštenja kod Progressive Web Aplikacija. Rad uključuje izradu svih potrebnih aplikacija za rešenje i demonstraciju na primeru aplikacije za anktetiranje. Rešenje se oslanja na korišćenje Service Worker-a i Web Push protokola.

2. TEHNOLOGIJE I EVOLUCIJA WEB APLIKACIJA

World Wide Web je izumeo Tim Berners-Lee 1989. godine. Berners-Lee je bio kompjuterski naučnik koji je radio u CERN-u, Evropskoj organizaciji za nuklearna istraživanja, kada je predložio sistem za razmenu informacija preko Interneta. Berners-Lee-jeva vizija je bila stvaranje decentralizovanog sistema za deljenje informacija, gde bi dokumenti mogli da budu povezani jedni sa drugima pomoću hiperlinkova. Izmislio je HTML (Hypertext Markup Language), koji je obezbedio standardizovan način formatiranja i strukturiranja Web stranica, i HTTP (Hypertext Transfer Protocol), koji je obezbedio standard za prenos podataka preko Interneta. Da bi demonstrirao potencijal Web-a, Berners-Lee je napravio prvu Web stranicu, koja je sadržala informacije o samom Web-u. Takođe je napravio prvi Web pretraživač, WorldWideWeb, koji je korisnicima omogućio pristup i navigaciju po mreži.

Web je brzo stekao popularnost, a do sredine 1990-ih, Web pretraživači i Web serveri su bili široko dostupni, što je svakome omogućilo da kreira i deli Web stranice. Ovo je označilo početak ere modernog Web-a, koji je od tada evoluirao u ogromnu mrežu međusobno povezanih Web stranica, aplikacija i servisa koje poznajemo danas.

Evolucija Web tehnologija može se široko kategorisati u tri faze:

1. Rani Web (Web 1.0) koja se takođe naziva statički Web ili read only Web je doba (period do sredine 2000-ih) kada je uloga korisnika ograničena na čitanje informacija koje pružaju proizvođači sadržaja. Ne postoji mogućnost da korisnik ili potrošač interaguju nazad ka serveru sa povratnim infrmacijama. Primer Weba 1.0 su statični Web sajtovi. Rani Web je prvenstveno napravljen korišćenjem statičkih HTML stranica, sa ograničenom interaktivnošću i funkcionalnošću. Uvođenje dinamičkog HTML-a (DHTML), JavaScript-a i jezika za skriptovanje na strani servera, kao što su PHP i ASP.NET, postavilo je osnovu za dinamičnije i interaktivnije Web aplikacije.

2. Web 2.0 je faza koju je obeležila pojava društvenih mreža i sadržaja koji generišu korisnici, kao i popularizacija Web 2.0 tehnologija kao što je AJAX, što je omogućilo asinhrona ažuriranja i dinamičnije korisničko iskustvo. Neke od poznatih Web 2.0 aplikacija su Facebook, Youtube, Flickr, Twitter itd. Web tehnologije poput HTML5, CSS3 i JavaScript framework-a kao što su ReactJs, AngularJs, VueJs itd., omogućavaju startapima da brzo razvijaju aplikacije koje omogućavaju interaktivno iskustvo sa korisnicima.

3. Web 3.0 koji se takođe naziva semantički Web ili čitanje-upisivanje-izvršavanje je era (2010. i novije) koja se odnosi na budućnost Web-a. U ovoj eri kompjuteri mogu tumačiti informacije poput ljudi putem veštačke inteligencije i mašinskog učenja. Koje pomažu da se inteligentno generiše i distribuira koristan sadržaj prilagođen posebnim potrebama korisnika. Neki od primera Web 3.0 aplikacija su Apple Siri, Google Cloud API, Wolfarm Aplha.

2.1 Web 1.0 - rana faza World Wide Web-a

Web 1.0 tehnologije se odnose na ranu fazu World Wide Web-a, od kasnih osamdesetih do sredine dvehiljdatiih. Neke od glavnih karakteristika, protokola i tehnologija koje su proizišle u ovom periodu uključuju:

1. HTTP (Hypertext Transfer Protocol) je protokol koji se koristi za prenos podataka preko interneta. To je osnova za komunikaciju podataka putem World Wide Web-a i koristi se za prenos informacija između Web pretraživača i servera. Kada korisnik unese URL u svoj Web pretraživač, HTTP zahtev se šalje serveru koji hostuje Web sajt. Server obrađuje zahtev i šalje nazad HTTP odgovor, koji uključuje sadržaj stranice i sve dodatne informacije potrebne za pravilno prikazivanje. HTTP je protokol bez stanja, što znači da ne održava vezu između klijenta i servera nakon što se svaki zahtev ispuni. Ovo omogućava veću fleksibilnost i skalabilnost, pošto su serveri u stanju da obrađuju mnogo zahteva istovremeno, bez potrebe za perzistentnim vezama.

2. HTTPS (Hipertekt Transfer Protocol Secure) je bezbedna verzija HTTP protokola koji se koristi za prenos podataka preko interneta. On šifruje podatke koji se prenose između klijenta i servera, obezbeđujući privatnost i sigurnost za osetljive informacije kao što su lozinke, brojevi kreditnih kartica i lične informacije. HTTPS koristi SSL (Secure Socket Layer) ili njegovog naslednika, TLS (Transport Layer Security), za šifrovanje podataka koji se prenose. SSL/TLS sertifikat, koji izdaje pouzdani autoritet za izdavanje sertifikata, koristi se za uspostavljanje bezbedne veze između klijenta i servera.

3. SMTP (Simple Mail Transfer Protocol) - koristi se za slanje i primanje e-poruka.

4. FTP i Telnet: FTP (File Transfer Protocol) i Telnet su korišćeni za prenos datoteka između servera i klijenata, kao i za daljinski pristup serverima i upravljanje sadržajem.

5. POP (Post Office Protocol) - koristi se za preuzimanje e-poruka sa servera.

6. IMAP (Internet Message Access Protocol) - koristi se za upravljanje porukama e-pošte na serveru.

7. HTML (Hypertext Markup Language): Standardni jezik za označavanje koji se koristi za kreiranje Web stranica i drugih informacija koje se mogu prikazati u Web pretraživaču. Koristi se za strukturiranje sadržaja i pružanje osnovnog izgleda za Web stranice, uključujući tekst, slike, veze i druge medije. HTML koristi niz oznaka i atributa da definiše strukturu i sadržaj Web stranice. Na primer, oznaka <p> se koristi za definisanje pasusa teksta, oznaka <img> se koristi za prikaz slika, a oznaka <a> se koristi za kreiranje hyperlinka (veze) ka drugim stranicama ili resursima. HTML ne pruža nikakve opcije za stilizovanje ili formatiranje. Stilom se obično rukuje pomoću CSS-a (Cascading Style Sheets), koji obezbeđuju način da se odvoji prezentacija Web stranice od njenog sadržaja.

8. Statički HTML: Većina Web sajtova tokom ove ere je napravljena pomoću statičkog HTML-a, što je značilo da sadržaj stranice nije mogao da se menja bez ručnog ažuriranja koda.

9. Rani Web pretraživači: Rani Web pretraživači, kao što su Mosaic i Netscape Navigator, korišćeni su za pristup i pregled Web stranica. Ovi pretraživači su olakšali navigaciju Web-om i gledanje multimedijalnog sadržaja.

10. Ograničena interakcija: Web 1.0 sajtovi su nudili ograničenu interaktivnost, pošto nije bilo podrške za dinamička ažuriranja ili korisnički unos. Web stranice su u suštini bile samo za čitanje, sa malo mogućnosti da korisnici komuniciraju sa sadržajem.

11. Bazični Web serveri: Web serveri, kao što je Apache, korišćeni su za hostovanje i serviranje Web stranica korisnicima.

U Web 1.0 eri nije bilo mnogo dostupnih Web framework-a za pravljenje Web aplikacija. Većina Web sajtova napravljena je korišćenjem jednostavnih, statičnih HTML stranica, sa ograničenom interakcijom i funkcionalnošću. Međutim, postojalo je nekoliko popularnih framework-a za Web razvoj koji su korišćeni u to vreme, uključujući:

- Common Gateway Interface (CGI): CGI je bio jedan od najranijih standarda za kreiranje dinamičkih Web stranica. Korišćen je za izvršavanje skripti na serveru i generisanje dinamičkog sadržaja, kao što su rezultati podnošenja formi ili sadržaj upita baze podataka. CGI skripte mogu biti napisane na različitim programskim jezicima. Skriptu izvršava Web server kad god korisnik zatraži dinamičku stranicu, a izlaz skripte se šalje nazad u pretraživač korisnika kao Web stranica.

- Active Server Pages (ASP.NET): ASP je Microsoft tehnologija koja je omogućava programerima da kreiraju dinamičke Web stranice koristeći skriptovanje na strani servera. Pruža framework lak za korišćenje za pravljenje jednostavnih Web aplikacija i široko je korišćen u ranim danima Weba. ASP nudi brojne funkcije i alate za izgradnju robusnih, skalabilnih i bezbednih Web aplikacija, uključujući:

    1. Model-View-Controller (MVC) arhitekturu: ASP.NET obezbeđuje ugrađenu MVC arhitekturu, koja olakšava izgradnju i održavanje složenih, dinamičnih Web aplikacija.

    2. Skriptovanje na strani servera: ASP.NET pruža bogat skup mogućnosti za skriptovanje na strani servera, uključujući podršku za različite programske jezike, kao što su C# i VB.NET.

    3. Pristup podacima: ASP.NET pruža ugrađenu podršku za pristup podacima iz različitih izvora, uključujući relacione baze podataka, XML i druge izvore podataka.

4. Bezbednost: ASP.NET uključuje brojne bezbednosne funkcije za zaštitu od uobičajenih bezbednosnih pretnji Web aplikacija, kao što su cross-site skriptovanje (XSS) i cross-site zabrana zahteva (CSRF).

Preduzeća i organizacije svih veličina naširoko koriste ASP.NET koji ima veliku i aktivnu zajednicu programera. To je moćan, fleksibilan i jednostavan framework za pravljenje dinamičkih Web aplikacija.

- PHP: PHP (Hypertext Preprocessor) je skriptni jezik na strani servera koji je dizajniran za pravljenje dinamičkih Web stranica. Prvi put je objavljen 1995. godine i brzo je postao popularan zbog svoje lakoće korišćenja i podrške za širok spektar baza podataka i Web servera. On je open-source, fleksibilan i efikasan, što ga čini popularnim izborom za pravljenje dinamičkih Web sajtova i aplikacija. PHP se može integrisati sa različitim bazama podataka kao što su MySQL, Oracle i PostgreSQL, i podržava različite protokole kao što su HTTP, HTTPS i SMTP. Kompatibilan je sa nizom operativnih sistema, uključujući Windows, Linux, and macOS, i može se pokrenuti na raznim Web serverima kao što su Apache, IIS i Nginx. Pored toga, PHP ima veliku zajednicu programera i ogromnu biblioteku modula i ekstenzija koje se mogu koristiti za proširenje njegove funkcionalnosti.

Uprkos svojim ograničenjima, Web 1.0 tehnologije i framework-ovi su odigrali ključnu ulogu u postavljanju temelja za sofisticiranije i interaktivnije Web aplikacije koje imamo danas. Rani Web otvorio je put razvoju novih tehnologija i framework-a, kao što su dinamički HTML (DHTML), JavaScript i skriptni jezici na strani servera, koji će se kasnije koristiti za izgradnju sofisticiranijih i dinamičnijih Web aplikacija.

2.2 Web 2.0 – era društvenih mreža

World Wide Web je doživeo ogromnu transformaciju od svog nastanka 1990-ih. Statički Web zasnovan na prikazivanju informacija evoluirao je u dinamičnu, interaktivnu platformu na kojoj korisnici mogu da učestvuju, sarađuju i kreiraju sadržaj. Ova evolucija se obično naziva Web 2.0. era. Ova era se često naziva erom Društvenih mreža koja olakšava interakciju između korisnika i sajtova koji interno omogućavaju korisnicima da komuniciraju međusobno.

Definišuće karakteristike Web-a 2.0 uključuju:

1. Društvene mreže: Pojava platformi društvenih mreža kao što su Facebook, Twitter, Youtube i LinkedIn imao je dubok uticaj na način na koji ljudi komuniciraju i dele informacije. Društveni mediji omogućavaju korisnicima da se povežu sa drugima, dele informacije i učestvuju u onlajn zajednicama.

2. Računarstvo u cloud-u: Uspon računarstva u oblaku omogućio je korisnicima pristup i čuvanje informacija na udaljenim serverima, a ne na njihovim ličnim računarima. Ovo je omogućilo korisnicima da pristupe svojim informacijama sa bilo kog mesta, u bilo koje vreme.

3. Sadržaj koji generišu korisnici: Web 2.0 je stavio veći naglasak na sadržaj koji generišu korisnici, kao što su blogovi, platforme za deljenje audio i video sadržaja i forumi za diskusiju. Ovo je osnažilo korisnike da učestvuju u kreiranju i distribuciji informacija.

Web 2.0 tehnologije i framework-ovi su gradivni blokovi koji su omogućili evoluciju World Wide Weba od jednostavne, statične platforme do dinamičke, interaktivne. Neke od najvažnijih Web 2.0 tehnologija uključuju:

1. JavaScript: JavaScript je skriptni jezik visokog nivoa na strani klijenta koji se koristi za kreiranje dinamičkih i interaktivnih Web stranica. To je suštinska komponenta Web 2.0 i podržavaju ga svi moderni Web pretraživači. JavaScript je jezik sa slabim tipovima podataka, interpretiran jezik koji je lako naučiti i često se koristi u kombinaciji sa HTML-om i CSS-om za kreiranje složenih i dinamičnih korisničkih interfejsa. Neke od ključnih karakteristika i prednosti JavaScript-a uključuju:

   - Dinamičku interaktivnost: JavaScript omogućava programerima da kreiraju dinamičke i interaktivne Web stranice, omogućavajući ažuriranje sadržaja, reagovanje na unos korisnika i kreiranje animacija i drugih efekata.
   - Cross-Platform: JavaScript podržavaju svi moderni Web pretraživači, što ga čini jezikom koji se može koristiti za pravljenje Web aplikacija koje rade na bilo kom uređaju.
   - Velika zajednica: JavaScript ima veliku i aktivnu zajednicu programera, pružajući obilje resursa, alata i podrške.
   - Kompatibilnost: JavaScript se može koristiti sa drugim tehnologijama, kao što su HTML i CSS, što omogućava pravljenje složenih i dinamičnih Web aplikacija.

   JavaScript je postao nezamenljiv alat za Web razvoj i koristi se za pravljenje širokog spektra aplikacija, od jednostavnih Web stranica do složenih Web aplikacija. Kako Web nastavlja da se razvija, JavaScript će nastaviti da igra ključnu ulogu u razvoju dinamičkih i interaktivnih Web stranica.

2. Ajax: Ajax (Asynchronous JavaScript and XML) je skup tehnologija koji omogućava Web stranicama da se dinamički ažuriraju bez ponovnog učitavanja cele stranice. Ovo je omogućilo stvaranje interaktivnijih i prilagodljivih Web aplikacija. Ajax funkcioniše tako što koristi JavaScript da kreiranje asinhronih zahteva ka serveru, preuzima podatke i ažurira deo Web stranice bez ponovnog učitavanja cele stranice (ažuriranjem DOM-a). Ovo omogućava Web aplikacijama da obezbede brže i responsivnije korisničko iskustvo. Korišćenjem Ajax-a se znatno smanjuje protok podataka, jer se uz pomoć njega mogu pribaviti samo podaci neophodni za ažuriranje dela strane umesto preuzimanja nove modifikovane stranice. Na taj način se dobijaju bolje performanse i manji troškovi protoka.

3. HTML5 i CSS3: HTML5 i CSS3 su najnovije verzije jezika za označavanje hiperteksta i kaskadnih stilova. Oni pružaju nove funkcije i mogućnosti za pravljenje dinamičkih Web stranica, a podržavaju ih svi moderni Web pretraživači.

4. Računarstvo u oblaku: Računarstvo u oblaku je model za isporuku računarskih resursa i servisa preko interneta. Omogućava programerima da prave i pokreću aplikacije na udaljenim serverima i obezbeđuje skalabilnost i ekonomičnost za Web aplikacije.

Tokom Web 2.0 ere pojavilo se puno biblioteka i framework-a koje su obezbedile brži i kvalitetniji razvoj klijentskih i serverskih Web aplikacija. Odabir tehnologije zavisi od potrebe same aplikacije, neke biblioteke su pogodnije za jednostavnije prezentacione aplikacije a druge

za obimne enterprise aplikacije.

Neki od najpopularnijih serverskih framework-ova:

1. Ruby on Rails: Framework visokog nivoa, full-stack napisan u Ruby-ju. Poznat je po svom pristupu „konvencija iznad konfiguracije“, što ga čini jednostavnim za kreiranje Web aplikacija vrlo brzo.

2. Django: Framework visokog nivoa, full-stack napisan u Python-u. Poznat je po svojoj sposobnosti da rukuje složenim aplikacijama i fokusu na bezbednost i skalabilnost.

3. Laravel: Framework visokog nivoa, full-stack napisan u PHP-u. Poznat je po svojoj elegantnoj sintaksi, jednostavnosti korišćenja i podršci za uobičajene zadatke Web razvoja.

4. Express: Minimalistički, brz i fleksibilan framework za pravljenje Web aplikacija korišćenjem Node.js-a.

5. Spring: Framework zasnovan na Javi za izradu enterprise Web aplikacija, poznat po svojim moćnim karakteristikama i robusnoj arhitekturi.

6. ASP.NET: Framework za pravljenje Web aplikacija i servisa sa .NET-om, poznat po svojim performansama, bezbednosti i skalabilnosti.

7. Flask: Lagan, minimalistički i fleksibilan framework za pravljenje Web aplikacija sa Python-om.

Neki od najpopularnijih klijentskih framework-ova i biblioteka:

1. React: JavaScript biblioteka za pravljenje korisničkih interfejsa, poznata po virtuelnom DOM-u, performansama i sposobnostima za rukovanje složenim aplikacijama.

2. Angular: JavaScript framework za izradu dinamičkih, složenih i skalabilnih aplikacija, poznat po svom dvosmernom povezivanju podataka, direktivama i podršci za komponente koje se mogu ponovo koristiti.

3. Vue.js: JavaScript framework za pravljenje korisničkih interfejsa, poznat po svojoj jednostavnosti, reaktivnosti i svestranosti.

4. Ember.js: JavaScript okvir za pravljenje ambicioznih Web aplikacija, poznat po svojim konvencijama i podršci za efikasan i održiv kod.

5. Backbone.js: minimalistička JavaScript biblioteka za pravljenje Web aplikacija, poznata po svojoj fleksibilnosti i podršci za modularnu event-driven arhitekturu.

6. Bootstrap: Popularan front-end framework za pravljenje Web-sajtova i Web aplikacija koje se prilagođavaju mobilnim uređajima, poznato po svojim komponentama korisničkog interfejsa, sistemu mreže i unapred dizajniranim temama.

Web 2.5 je koncept koji se bavi praktičnom i stvarnom evolucijom između Web-a 2.0 i 3.0. Na putu ka Web 3.0 neki od kompanija kao što su Amazon, Google, Microsoft itd, pružaju model servisa (usluge) u računarstvu u oblaku, omogućavajući lak razvoj Web aplikacija koje se mogu pokrenuti sa bilo kog uređaja, bilo kada i bilo gde.

Računarstvo u oblaku je model za pružanje usluga informacionih tehnologija preko interneta. Umesto pokretanja softvera i skladištenja podataka na ličnim računarima ili lokalnim serverima, korisnici mogu pristupiti ovim resursima preko interneta iz udaljenih centara podataka. Ovo omogućava veću skalabilnost, fleksibilnost i ekonomičnost u poređenju sa tradicionalnim IT rešenjima. Postoje tri glavna tipa usluga računarstva u oblaku:

1. Infrastruktura kao servis (IaaS): Obezbeđuje virtuelizovane računarske resurse, uključujući servere, skladište i umrežavanje, preko interneta.

2. Platforma kao servis (PaaS): Obezbeđuje platformu za izgradnju, primenu i upravljanje Web aplikacijama i servisima, bez potrebe za upravljanjem osnovnom infrastrukturom.

3. Softver kao servis (SaaS): Omogućava pristup softverskim aplikacijama i alatima preko interneta, eliminišući potrebu za njihovim instaliranjem i pokretanjem na ličnim računarima.

Prednosti računarstva u oblaku uključuju:

- Uštedu: Korisnici plaćaju samo za resurse koje koriste, umesto da moraju da ulažu u skupi hardver i softver.
- Skalabilnost: Resursi se mogu povećati ili smanjiti prema potrebi da bi se ispunili promenljivi zahtevi.
- Pristupačnost: Aplikacijama i podacima se može pristupiti sa bilo kog mesta sa internet vezom.
- Pouzdanost: Provajderi u oblaku često imaju više centara podataka i rezervnih sistema kako bi osigurali visoku dostupnost i pouzdanost.

Web 2.5 se uglavnom fokusira na mobilno računarstvo i evoluciju mobilnih tehnologija. Kako mobilno računarstvo igra bitnu ulogu u privlačenju većeg broja korisnika putem native aplikacija, kao i mobilnih Web aplikacija, možemo pronaći sve više i više aplikacija koje ulaze na tržište mobilnih uređaja kako bi uspostavile svoje prisustvo obraćajući se mobilnim korisnicima. Neke od tehnologija predstavljenih u ovoj eri Web-a uključuju Progressive Web Aplikacije (PWA), Accelerated Mobile Pages (AMP) i druge.

2.3 Web 3.0 – semantički Web

Koncept Web-a 3.0, takođe poznat kao semantički Web, predstavlja sledeću fazu u evoluciji interneta. Predstavlja pomak sa sadašnjeg modela Web-a, koji je fokusiran na isporuku informacija i sadržaja korisnicima, na model koji se zasniva na inteligenciji i mašinski čitljivim podacima. Ima za cilj da stvori povezaniji i inteligentniji Web tako što će omogućiti stvaranje zajedničkog, decentralizovanog i mašinski čitljivog sloja podataka kojem mogu pristupiti i razumeti i ljudi i mašine. Ovo se postiže korišćenjem tehnologija kao što su povezani podaci, ontologije i veštačka inteligencija.

Ključne karakteristike Web-a 3.0 uključuju:

- Decentralizacija: Web 3.0 ima za cilj da se udalji od centralizovanog skladištenja podataka ka decentralizovanim sistemima, omogućavajući veću kontrolu i vlasništvo nad podacima od strane pojedinaca i organizacija.
- Interoperabilnost: Web 3.0 ima za cilj da omogući veću interoperabilnost između različitih sistema i platformi, omogućavajući besprekorniju razmenu informacija i podataka između aplikacija.
- Inteligencija: Web 3.0 ima za cilj da učini Web inteligentnijim omogućavanjem kreiranja aplikacija i sistema koji mogu da razumeju, obrađuju i razmišljaju o podacima i informacijama.
- Usredsređenost na korisnike: Web 3.0 ima za cilj da stavi korisnike u centar Weba dajući im veću kontrolu i vlasništvo nad njihovim podacima i omogućavajući im da komuniciraju sa Webom na nove i ličnije načine.

Web 3.0 ima potencijal da donese značajne promene u načinu na koji koristimo i komuniciramo sa Web-om. Omogućavajući veću kontrolu, vlasništvo i inteligenciju na Webu, on ima potencijal da stvori nove mogućnosti za inovacije i da transformiše način na koji radimo, učimo i komuniciramo. Međutim, razvoj i implementacija Web 3.0 tehnologija će zahtevati značajne tehničke i kulturne promene i zahtevaće saradnju širokog spektra zainteresovanih strana, uključujući istraživače, programere, donosioce zakona i korisnike.

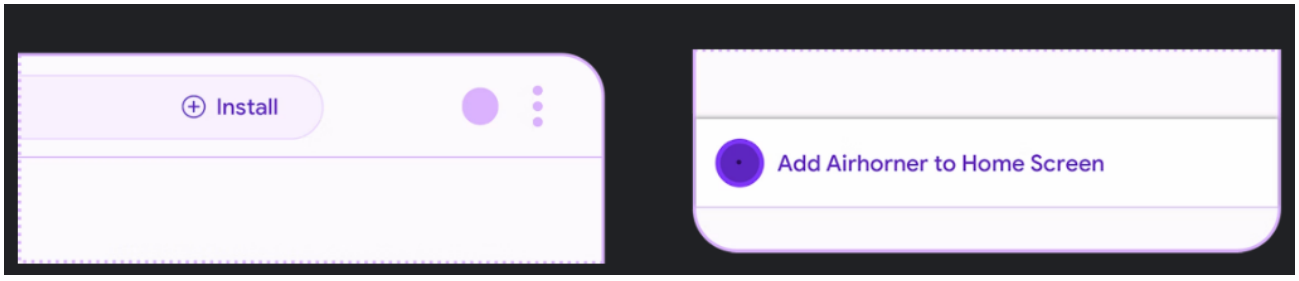
3. PROGRESSIVE WEB APPLIKACIJE

3.1 Karakteristike Progressive Web Aplikacija

Progressive Web aplikacije (PWA) su tip Web aplikacija koje pružaju korisničko iskustvo nalik native mobilnim aplikacijama, ali rade u Web pretraživaču. Instalirane Progressive Web aplikacije rade u samostalnom prozoru umesto na kartici pretraživača. Mogu se pokrenuti sa početnog ekrana korisnika, doka, ili sa trake sa obaveštenjima. Moguće je tražiti ih na uređaju i skakati između njih poput drugih aplikacija koje su instalirane na uređaju.

Nove mogućnosti se otvaraju nakon što se instalira Web aplikacija. Prečice na tastaturi, obično rezervisane kada se aplikacije pokreće u pretraživaču, postaju dostupne. Progressive Web aplikacije mogu da se registruju da prihvataju sadržaj iz drugih aplikacija ili da budu podrazumevana aplikacija za rukovanje različitim tipovima datoteka.

Kada se Progressive Web aplikacija pomeri sa kartice u prozor samostalne aplikacije, ona transformiše način na koji korisnici razmišljaju o njoj i komuniciraju sa njom. Većina pretraživača ukazuje korisniku da se Progressive Web aplikacija može instalirati kada ispunjava određene kriterijume. Na slici 1 se vide primeri indikatora uključujući dugme Instaliraj na traci za adresu ili stavku menija Instaliraj u preklopnom meniju.



Slika 1. Primer dugmeta za instaliranje aplikacije

Jedna od ključnih karakteristika PWA a je offline funkcionalnost, mogu se konfigurisati da rade offline korišćenjem Service Worker-a, to su JavaScript datoteke koje se pokreću odvojeno od Web stranice i mogu da obrađuju mrežne zahteve i keširanje. Ovo omogućava aplikaciji da nastavi da funkcioniše, čak i kada je uređaj korisnika van mreže.

Osnovni tok PWA je sledeći:

1. Korisnik posećuje PWA URL koristeći moderni Web pretraživač.

2. Pretraživač od servera zahteva HTML, CSS i JavaScript datoteke koje čine PWA.

3. Pretraživač pokreće JavaScript kod, koji koristi HTML i CSS da prikaže korisnički interfejs i odgovori na interakcije korisnika.

4. Ako korisnik odluči da instalira PWA, pretraživač ga dodaje na početni ekran uređaja, omogućavajući mu da se pokrene kao native mobilna aplikacija.

5. PWA koristi Service Worker-e da obezbedi offline funkcionalnost, pozadinska ažuriranja i push obaveštenja, čineći aplikaciju privlačnijom i brzom, čak i u oblastima sa malom pokrivenošću mrežom.

6. PWA takođe može da koristi API-je kao što su Cache API i Background Sync API za keširanje resursa aplikacija i izvršavanje ažuriranja u pozadini, dodatno poboljšavajući korisničko iskustvo.

Još jedna karakteristika PWA je mogućnost primanja push obaveštenja, koja se može implementirati korišćenjem Web push API-ja i usluge za razmenu poruka kao što je Firebase Cloud Messaging.
Service Workeri, Notifications API i Web Push Protocol su ključne tehnologije koje omogućavaju PWA-ovima da obezbede iskustvo poput native mobilnih aplikacija na Webu. Korišćenjem ovih tehnologija, PWA mogu da obezbede brzo, responsivno i privlačno korisničko iskustvo i da ponude isplativo i skalabilno rešenje za preduzeća i organizacije koje žele da dopru do šire publike i obezbede bolje korisničko iskustvo na Webu.

Web Push protokol se koristi za slanje push obaveštenja korisnicima, čak i kada PWA nije pokrenuta. Ovo omogućava PWA da obezbede blagovremena i relevantna ažuriranja i obaveštenja za korisnika, bez potrebe za native aplikacijom.

Service Worker-i su vrsta skripte koja se pokreće u pozadini pretraživača i može da rukuje mrežnim zahtevima, kešom i pruža offline funkcionalnost.

Notifications API omogućava PWA da prikažu obaveštenja korisniku, čak i kada PWA nije aktivno pokrenut. Ovo omogućava PWA-ima da pruže zanimljivije iskustvo i obaveštavaju korisnika o novim događajima ili ažuriranjima.

3.2 Web Push protokol i Push API

Web Push protokol je skup pravila i smernica za slanje push obaveštenja Web pretraživaču, dok je Push API API pretraživača koji omogućava Web aplikacijama da primaju push obaveštenja od push servisa.

Web Push protokol koristi standardni HTTP/2 protokol za slanje i primanje poruka i zahteva upotrebu push servisa za rukovanje slanjem i prijemom push poruka. Push API, sa druge strane, je JavaScript API koji omogućava Web aplikacijama da se pretplate na push obaveštenja, primaju push poruke i upravljaju prikazom obaveštenja.

Server može da pošalje push poruku u bilo kom trenutku, čak i kada je Web aplikacija ili korisnički agent neaktivan. Push servis osigurava pouzdanu i efikasnu isporuku korisničkom agentu. Push poruke se isporučuju Service Worker-u koji se pokreće u okviru Web aplikacije, koji može da koristi informacije u poruci da ažurira lokalno stanje ili prikaže obaveštenje korisniku. Push poruke se čuvaju na push servisu sve dok agent ne postane dostupan i poruka ne bude isporučena.
Push poruke je najbolje koristiti kada ne postoji direktna komunikacija između agenta i Web aplikacije. Slanje push poruka zahteva znatno više resursa u poređenju sa direktnim metodama komunikacije poput REST poziva ili WebSocket-a, i latencije su obično veće. Većina push servisa ograničava veličinu i količinu push poruka koje mogu biti poslate.

Web push protokol je zasnovan na IETF HTTP/2 specifikaciji i koristi sledeće entitete za slanje i primanje push obaveštenja:

1. Application server: Odnosi se na komponente aplikacije na strani servera.

2. User agent - Ovo je komponenta na strani klijenta koja prima push obaveštenja. To može biti Web pretraživač ili PWA.

3. Push Service: Posrednik između application servera i user agenta.

4. Push Subscription: Ovo je objekat koji sadrži informacije potrebne push service-u da usmeri push obaveštenje do odgovarajućeg korisničkog agenta.

5. Push Message: Sadržaj obaveštenja, sadržaj poruke koji može biti string ili niz bajtova, meta podatke kao što su naslov poruke ili ikonica.

6. VAPID (Voluntary Application Server Identification): Mehanizam koji omogućava application server-u da se identifikuje na push servisu. Sadrži javni i privatni ključ, koji se koristi za potpisivanje JWT tokena koji se šalje sa push porukom.

7. Endpoint: URL push servisa gde se čuvaju push subscription-i.

8. P256DH Key: Javni ključ koji se koristi za šifrovanje push poruke pre slanja user agentu.

9. Auth Key: Privatni ključ koji se koristi za proveru integriteta poruke kada stigne user agentu.

3.3 Service Worker-i

Service Worker je skripta koja se pokreće u pozadini Web pretraživača, odvojeno od Web stranice, i deluje kao proksi između Web stranice i mreže. Omogućava funkcije kao što su podrška offline moda rada, sinhronizacija podataka u pozadini i push obaveštenja. Imaju mogućnost presretanja i rukovanja HTTP pozivima, uključujući keširanje odgovora. Ovo omogućava Web stranicama i Web aplikacijama da rade u offline modu i poboljšava performanse učitavanja Web stranica.

Service Worker je u stanju da presreće HTTP zahteve, tako da može da odluči da li da odgovori sa keširanom verzijom resursa ili da izvrši poziv kako bi dobio podatke. Takođe može da upravlja push obaveštenjima i sinhronizacijom u pozadini. To su događaji koji ne zahtevaju otvorenu karticu pretraživača da bi funkcionisali.

Service Worker-a registruje Web stranica i on može da kontroliše sve stranice koje se učitavaju sa istog domena, što je lokacija datoteke Service Worker-a. Jednom kada je Service Worker registrovan i instaliran, on ostaje aktivan sve dok ga ne zameni novi Worker ili ga ne odjavi Web stranica.

Prenos podataka preko mreže je inherentno asinhroni. Potrebno je vreme da se zahteva resurs, da server odgovori na taj zahtev i da se odgovor preuzme. Potrebno vreme je raznoliko i neodređeno.
Service Worker-i prihvataju ovu asinhronost preko event-driven API-ja koristeći callback-ove za event-e. Neki od event-a su: activate, install, fetch, message, push.

Service Worker-i se izvršavaju u posebnim, tj. sopstvenim nitima i na taj način ne opterećuju glavnu nit aplikacije.

Važno je napomenuti da Service Worker nije podržan od strane svih pretraživača, trenutno ga podržava većina modernih pretraživača, ali za neke stare pretraživače možda nije podržan.

Service Worker predstavlja korisničkog agenta u okviru Web push protokola.

Service Worker-i imlementiraju nekoliko metoda koje se mogu koristiti za izvršavanje različitih zadataka. Neke od ključnih metoda su:

- „self.register()“: Ova metoda se koristi za registrovanje. Prosleđuje joj se parametar koji je URL do datoteke worker-a i vraća promise koji se razrešava „ServiceWorkerRegistration“ objektom.
- „self.skipWaiting()“: Ova metoda omogućava workeru da preuzme kontrolu od postojećeg worker-a na stranici odman nakon instalacije, tj. zaobilazi čekanje na zatvaranje svih otvorenih kartica pretraživača.
- „self.unregister()“: Metoda koja se koristi za odjavu workera.
- „self.addEventListener()“: Ova metoda se koristi za dodavanje osluškivača događaja na worker.
- „cache“ metode za manipulaciju nad kešom.

Postoji nekoliko događaja koji se mogu iskoristiti za izvršavanje različitih zadataka, koje je moguće kontrolisati uz pomoć „self.addEventListener()“ metode:

- „install“: Ovaj događaj se okida kada se Service Worker prvi put instalira. Obično se koristi za keširanje asset-a i podešavanja inicijalnog stanja workera.
- „activate“: Ovaj događaj se okida nakon što se worker instalira i preuzme kontrolu nad stranicom. Može se koristiti za čišćenje starih keša ili za obavljanje drugih zadataka koje treba obaviti pre nego što worker počne da obrađuje zahteve.
- „fetch“: Ovaj događaj se okida svaki put kada se uputi HTTP zahtev sa stranice koju kontroliše serviser. Može se koristiti za presretanje zahteva i posluživanje keširanih odgovora ili obavljanje drugih zadataka u vezi sa zahtevom.
- „message“: Ovaj događaj se okida kada worker primi poruku sa stranice ili drugog servisera. Može se koristiti za komunikaciju između worker-a i Web aplikacije ili između različitih worker-a.
- „push“: Ovaj događaj se okida kada worker primi push obaveštenje. Može se koristiti za prikazivanje obaveštenja korisniku ili za obavljanje drugih zadataka u vezi sa push obaveštenjem.
- „sync“: Ovaj događaj se okida kada se događaj sinhronizacije registruje pomoću self.registration.sinc.register('tagName') i koristi se za obavljanje pozadinskih zadataka, kao što je slanje podataka serveru ili ažuriranje sadržaja aplikacije, čak i kada je aplikacija zatvorena.
- „notificationclick“: Ovaj događaj se okida kada korisnik stupi u interakciju sa obaveštenjem. Može se koristiti za otvaranje određene stranice ili obavljanje drugih zadataka u vezi sa obaveštenjem.

3.4 Notifications API

Notifications API je Web tehnologija koja omogućava Web aplikacijama da prikazuju push obaveštenja korisniku, čak i kada aplikacija nije aktivno pokrenuta. Ovaj API pruža način da Web aplikacije obezbede zanimljivije i informativnije korisničko iskustvo i može se koristiti za obaveštavanje korisnika o novim događajima ili ažuriranjima, pružanje podsetnika i još mnogo toga.

Notifications API radi tako što šalje zahtev pretraživaču da prikaže push obaveštenje korisniku. Zahtev sadrži informacije kao što su naslov, poruka i ikona za prikaz, kao i opcije za obaveštenje, kao što su zvuk ili vibracija koja će se koristiti.

Jedna od ključnih prednosti Notifications API-ja je njegova sposobnost da radi u pozadini, čak i kada aplikacija nije aktivno pokrenuta. To znači da Web aplikacije mogu obezbediti pravovremena i

relevantna ažuriranja za korisnika, bez potrebe za native aplikacijom. Pored mogućnosti da radi u pozadini, Notifications API je takođe jednostavan za korišćenje i široko podržan od strane modernih pretraživača, što ga čini popularnim izborom za Web programere. Štaviše, to je standardni deo Web platforme, što znači da je kompatibilan sa svim Web tehnologijama, uključujući i PWA.

Notifications API je podržan od strane većine modernih Web pretraživača, uključujući:

- Google Chrome (verzija 22 ili novija)
- Firefox (verzija 22 ili novija)
- Safari (verzija 6 ili novija)
- Opera (verzija 15 ili novija)
- Microsoft Edge (verzija 79 ili novija)
- Samsung Internet (verzija 4.0 ili novija)

Treba napomenuti da Notification API nije podržan na Internet Explorer-u.

Pored toga, mogućnost prikazivanja push obaveštenja na mobilnim platformama varira. Na iOS-u, push obaveštenja se prikazuju pomoću Apple Push Notification usluge (APNS), koja zahteva dodatno podešavanje i konfiguraciju.

4. ARHITEKTURA I IMPLEMENTACIJA SISTEMA ZA ANKETIRANJE KORISNIKA

4.1 Zadatak

Zadatak je kreirati demo aplikaciju koja demonstrira korišćenje gore pomenutih tehnologija.
Kao primer uzeta je tema mobilne aplikacije za anketiranje.

Aplikacija je zamišljena tako da registrovani korisnici mogu kreirati ankete koje sadrže više pitanja od 4 moguća tipa:

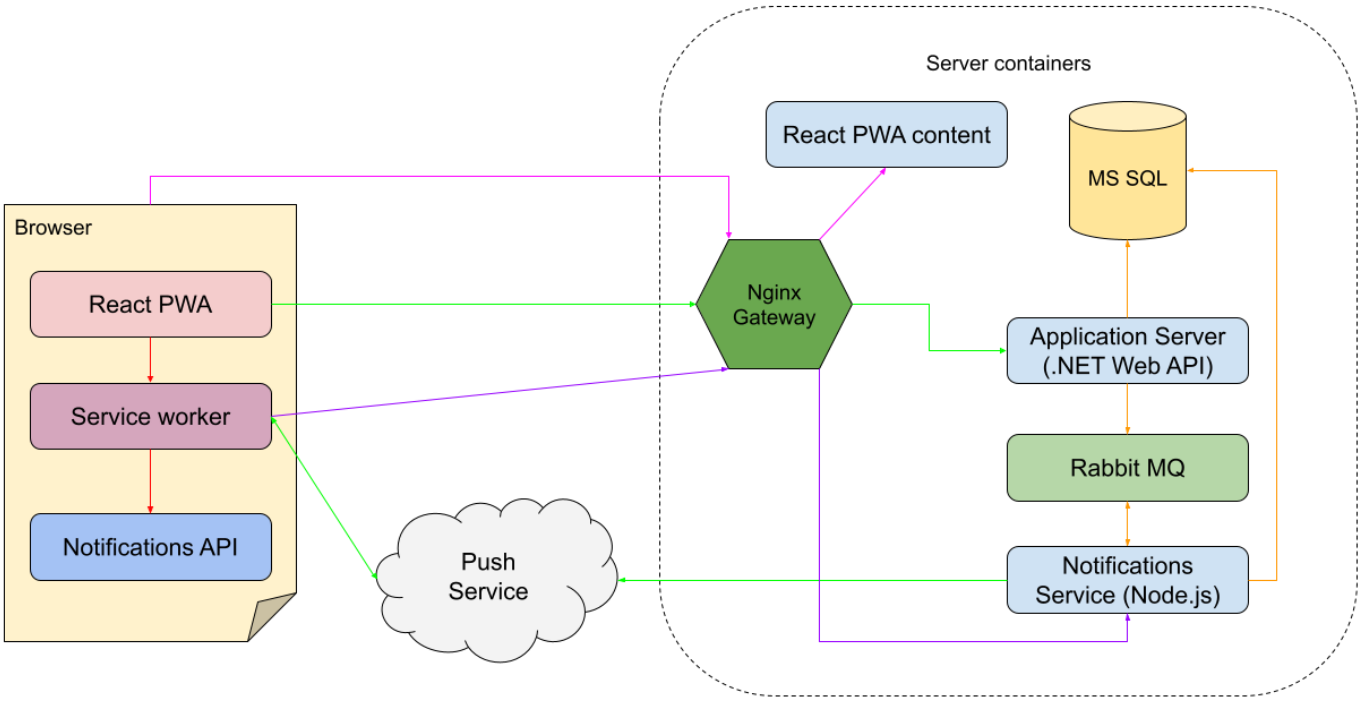
1. pitanje na koje je moguće odgovoriti sa da ili ne
2. pitanje na koje se odgovara jednim od ponuđenih opcija
3. pitanje na koje se odgovara jednom od više ponuđenih opcija
4. pitanje na koje se daje tekstualni odgovor.

Sve kreirane ankete se mogu prelistavati na početnoj strani aplikacije gde svi korisnici mogu odgovatati na iste. Kada neki korisnik kreirana novu anketu svim ostalim korisnicima se šalje push poruka koja ih obaveštava da postoji nova anketa koju je moguće odgovoriti, a kada neko pošalje odgovor na anketu, korisniku koji je vlasnik te ankete stiže push poruka koja ga obaveštava da ima novi odgovor. Postoji i stranica na kojoj je moguće videti sve odgovore na anketama koje je kreirao prijavljeni korisnik.

Implementacija podrazumeva kreiranje React PWA aplikacije koja se hostuje kao Web sajt, serverske aplikacije (.NET Web API) koja skladišti sve podatke u MS SQL bazu i implementaciju push poruka korišćenjem Web Push protokola, Push API-ja i Service Worker.

4.2 Arhitektura aplikacije

Na slici 2 se nalazi arhitektura aplikacije. Osnovnu arhitekturu aplikacije čine React PWA klijentska aplikacija, .NET Web API aplikacioni server i MS SQL baza podataka. Koristeći klijentsku aplikaciju korisnici mogu da daju odgovore na ankete, kreiraju svoje ankete i pogledaju analizu odgovora na svoje ankete. Aplikacioni server je RESTful .NET Web API sa svim potrebnim endpoint-ima za manipulaciju nad entitetima aplikacije.



Slika 2. Arhitektura aplikacije

Na serverskoj strani se pored aplikacionog servera nalazi i Notifications servis (Node.js aplikacija) koji je odgovoran za slanje obaveštenja, koje aplikacioni server doda na red za slanje, ka korisnicima. Aplikacioni server i Notifications servis komuniciraju preko redova na Rabbit MQ-u.

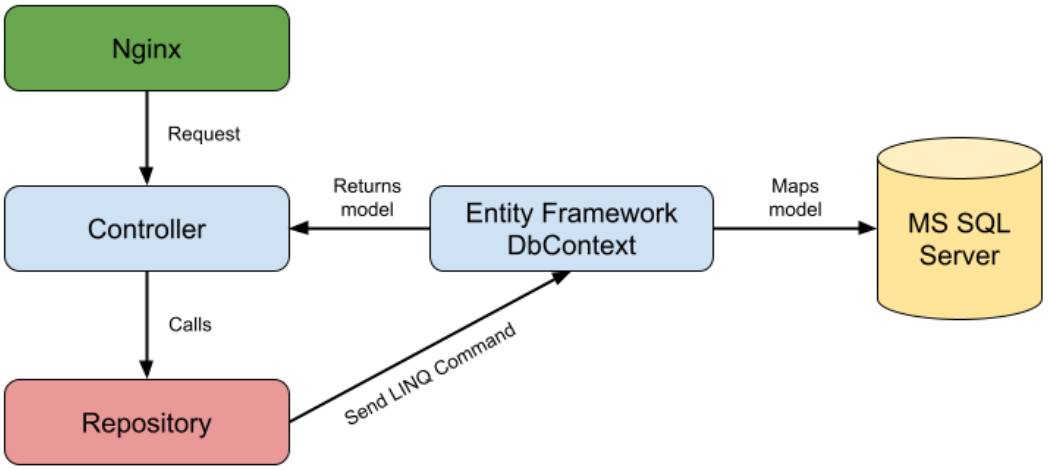
Kada se korisnik registruje i prijavi se unutar klijentske aplikacije, tada aplikacija registruje Service Worker file koji je odgovoran za rukovanje push porukama. Service Worker odmah po učitavanju kreira push subscription objekat pozivanjem odgovarajuće metode na Push Service-u. Nakon toga taj push subscription objekat uz identifikator korisnika šalje Notifikacionom servisu, jer kasnije na osnovu njega servis doprema poruke nazad do Service Workera za prikaz. Notifikacioni servis čuva push susbscription objekat unutar MS SQL baze. Kada desi bilo koji događaj unutar sistema koji kao rezultat ima obaveštenje, aplikacioni server dodaje na red poruku koja može biti namenjena jednom korisniku ili svim korisnicima. Notifikacioni servis prihvata poruke sa reda i na osnovu push subscribtion objekata emituje poruku na Push Service i on je doprema nazad do klijenta koji je prikazuje koristeći Notifications API ugrađen u pretraživač.

Na serveru se pored ostalih aplikacija nalazi i Nginx Gateway koji je neophodan zbog dodavanja SSL sertifikata.

4.3 Komponente aplikacije

4.3.1 Aplikacioni Server (.NET Web API i MS SQL server)

Server aplikacije zadužen je za registraciju i autorizaciju korisnika, dodavanje novih anketa u bazu, listanje anketa i sastavljanje pregleda odgovora. Server je napisan koristeći .NET framework, tj ASP.NET Web API framework. Za autentifikaciju i autorizaciju je iskorišćen ASP.NET Identity a za komunikaciju sa serverom baze podataka Entity Framework Core. Za organizaciju arhitekture servera (slika 3) je iskorišćen poznati Repository patern, koji se koristi za apstraktovanje sloja za pristup podacima od sloja poslovne logike aplikacije.



Slika 3. Arhitektura aplikacionog servera

4.3.1.1 ASP.NET Web API

ASP.NET Core je framework za pravljenje Web API-ja i Web aplikacija koristeći .NET framework. To je lightweight framework visokih performansi koji može da radi na različitim platformama, uključujući Windows, Linux i macOS.

Koristeći ASP.NET Core mogu se kreirati REST Web API-ji koji upravljaju HTTP zahtevima i odgovorima. Moguće je upravljati različitim tipovima HTTP zahteva kao što su: GET, POST, PUT i DELETE, takođe obezbeđuje upravljanje query i route parametrima i validaciju podataka.

ASP.NET Core uključuje ugrađeni dependency injection sistem koji olakšava upravljanje zavisnostima aplikacije, middleware pipeline koij omogućava dodavanje funkcionalnosti aplikacije u različitim fazama request-response ciklusa. ASP.NET Core podržava širok spektar funkcija kao što su: validacija modela, autentifikacija i autorizacija, keširanje, logiranje, upravljanje izuzecima, konfiguracija, lokalizacija, komunikacija u realnom vremenu koristeći WebSocket-e ili SignalR.
Web API je moguće host-ovati na različitim platformama uključujući IIS, Nginx, Apache.

API metode aplikacionog servera:

- POST metoda *„api/users/register“*: Metoda koja se poziva kada se registruje novi korisnik. U telu zahteva se kao parametri porleđuju korisničko ime, email adresa, puno ime korisnika i lozinka.
- POST metoda *„api/users/login“*: Metoda koja se poziva kada se korisnik prijavljuje, tj kada mu se izdaje access token. U telu zahteva se prosleđuju korisničko ime i lozinka.
- GET metoda *„api/polls“*: Lista sve kreirane ankete, kao query parametri se mogu proslediti filteri za paginaciju (offset i limit), string za pretragu „searchParam“ (ne razmatra se ako je prazan) i boolean parametar „getForUser“ koji određuje da li se ankete listaju za samo od prijavljenog korisnika ili od svih.
- GET metoda *„api/polls/{id}“*: Metoda koja se koristi za pribavljanje punog objekta ankete po identifikatoru ankete koji se nalazi u putanji zahteva.
- POST metoda *„api/polls“*: Metoda kojom se kreira nova anketa, u telu zahteva se prosleđuju ime i opis ankete i lista pitanja.
- POST metoda *„api/polls/{pollId}/questions“*: Metoda kojom se dodaju nova pitanja na postojeću anketu. Indentifikator ankete se prosleđuje kroz putanju zahteva a lista pitanja u telu zahteva.
- DELETE metoda *„api/polls/{pollId}/questions“*: Metoda kojom se brišu pitanja sa postojeće ankete. Indentifikator ankete se prosleđuje kroz putanju zahteva a lista indentifikatora pitanja koja se brišu u telu zahteva.
- DELETE metoda *„api/polls/{pollId}“*: Metoda kojom se briše anketa sa svim pitanjima i odgovorima. Indentifikator ankete se prosleđuje u putanji zahteva.
- POST metoda *„api/answers/{pollId}“*: Metoda kojom se dodaju odgovori na anketu. U putanji zahteva se prosleđuje indentifikator ankete a u telu zahteva odgovori.
- GET metoda *„api/answers/{pollId}“*: Metoda koja vraća listu svih odgovora na anketu.

Svi detalji API-ja zajedno sa Data transfer objektima se mogu videti kroz Swagger file koji je dostupan na adresi *„{URL_HOSTA}/api/swagger“*.

4.3.1.2 ASP.NET Core Identity

ASP.NET Core Identity je membership sistem koji omogućava dodavanje funkcionalnosti anutentifikacije i autorizacije u .NET Web aplikacijama. Pruža kompletan proširivi skup funkcionalnosti za upravljanje autentifikacijom i autorizacijom , uključujući: registraciju i prijavljivanje korisnika, autentifikacija zasnovana na email-u i lozinki, dvofaktorska autentifikacija, prijava preko društvenih mreža, kontrola pristupa zasnovana na rolama, autentifikacija korišćenjem tokena, oporavak lozinke, potvrda naloga i drugo.

ASP.NET Core Identiti skladišti korisničke podatke, uključujući kredencijale i informacije o profilu, u relacionoj bazi podataka kao što je MS SQL Server, MySQL ili PostgreSQL.

4.3.1.3 Entity Framework Core i Microsoft SQL Server

Entity Framework Core (EF Core) je open-source, lightweight verzija poularne Entitz Framework tehnologije pristupa podacima. EF Core može da se koristiti sa različitim bazama podataka, uključujući Microsoft SQL Server. Obezbeđuje framework objektno-relacionog mapiranja (ORM) za .NET aplikacije, omogućavajući rad rade sa bazom podataka koristeći objekte specifične za domen umesto pisanja SQL upita.

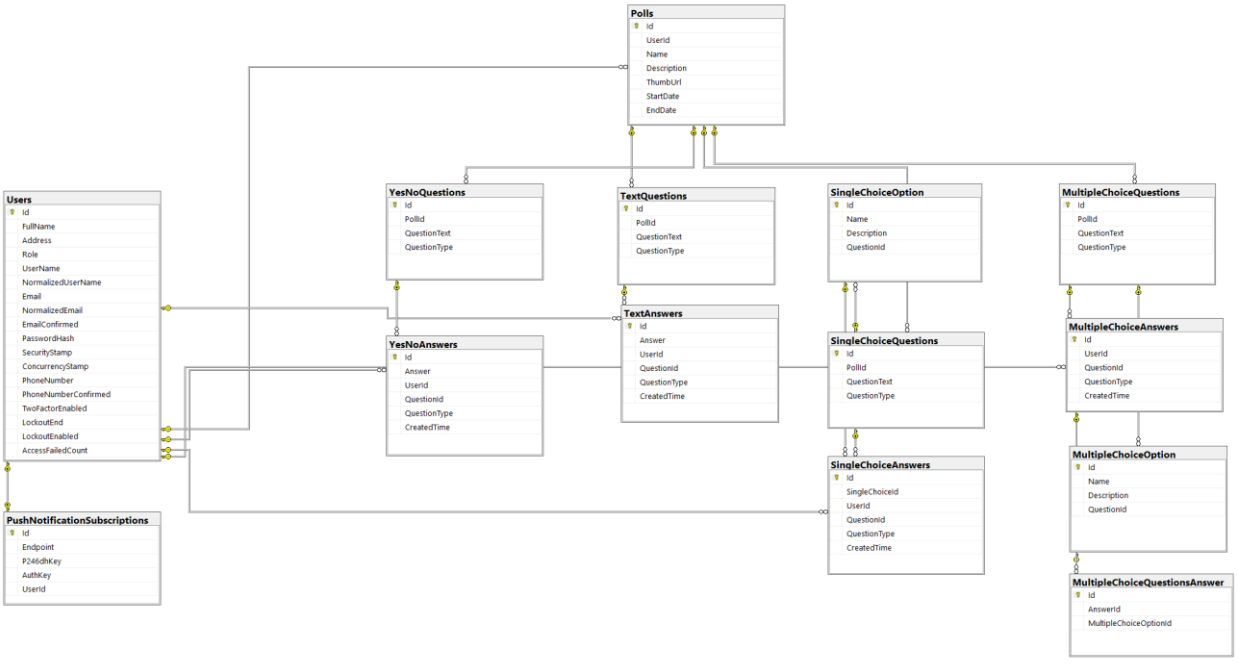
Kada se EF Core koristi sa MS SQL Serverom biblioteka obezbeđuje provajder baze podataka, što

omogućava infrastrukturi EF Core-a interakciju sa bazom podataka. Ovaj provajder je odgovoran za prevođenje operacija koje nad entitetima obavlja infrastruktura EF Core-a u odgovarajuće SQL upite koje može da izvrši SQL Server baza. EF Core se može koristiti za rukovanje zadacima kao što su kreiranje, čitanje, ažuriranje i brisanje zapisa u bazi podataka, a takođe omogućava ispitivanje podataka pomoću LINQ-a. Takođe pruža podršku za migracije baze podataka, što vam omogućava da razvijate šemu baze podataka tokom vremena kako se vaša aplikacija razvija.

Koristeći EF Core, moguće je pisanje koda koji ne zavisi od baze podataka, što znači da se može koristiti sa različitim bazama podataka uz minimalne promene. Ovo omogućava veću fleksibilnost u izboru tehnologije baze podataka i takođe olakšava promenu baze podataka u budućnosti.

U slučaju naše aplikacije korišćen je code first pristup, gde se prvo kreiraju klase domena (POCO objekti), na osnovu kojih EF Core kreira bazu podataka. Zatim se kreira „DbContext“ klasa koja nasledjuje „Microsoft.EntitzFrameworkCore.DbContext“ klasu, ona služi kao most između domenskih klasa i baze. Definiše se šema baze koristeći Data Annotation atribute ili Fluent API nad domenskim klasama i nad DbContext-om. Konfiguriše se konekcioni string. DbContext se koristi za izvšavanje operacija na d bazom podataka ko što su ispitivanje, dodavanje, ažuriranje i brisanje podataka. DbContext će automatski kreirati bazu ukoliko ona već ne postoji. Kada se menjaju domenske klase mogu se koristiti EF Core migracije za ažuriranje šeme baze podataka.

Microsoft SQL Server je sistem za upravljanje relacionim bazama podataka (RDBMS) koji je razvio i plasirao Microsoft. Koristi se za upravljanje i skladištenje podataka za različite aplikacije i podržava širok spektar tipova podataka i modela podataka, uključujući strukturirane i nestrukturirane podatke. Može se pokrenuti lokalno, u oblaku ili kao hibridno rešenje. Takođe pruža funkcije kao što su visoka dostupnost, skalabilnost i bezbednost.



Slika 4. Relaciona šema baze

Na slici 4 se nalazi relaciona šema baze podataka. Svaki korisnik (User) može kreirati više anketa (Polls), a svaka anketa može da ima više pitanja svakog od 4 moguća tipa (YesNoQuestion, TextQuestion, SingleChoiceQuestion, MultipleQuestion). Ponuđeni odgovori kod pitanja sa odabirom se čuvaju u SingleChoiceOption i MultipleChoiceOption tabelama. Za svaki od tipova pitanja postoji odgovarajuća tabela u kojoj se pamte odgovori.

4.3.2 Notifications Service

Notifications Service je Node.js aplikacija čiji je zadatak da prihvata push subscription objekte i sačuva ih uz odgovarajućeg korisnika u postojećoj MS SQL bazi i da emituje push obaveštenja, koja primi od strane aplikacionog servera, jednom korisniku ili grupi korisnika.

Node.js je open-source okruženje za izvršavanje JavaScript koda, koje podržava više platformi. Node se u velikoj meri koristi za programiranje na strani servera, što omogućava programerima da koriste JavaScript za klijentski i serverski kod bez potrebe da uče dodatni jezik. Koristi V8 JavaScript engine za izvršavanje koda, to je engine koji koristi i Google Chrome pretraživač.

Node.js pruža bogatu biblioteku modula kroz Node Package Manager (npm) koji olakšava instalaciju i korišćenje različitih biblioteka i framework-ova za obavljanje različitih zadataka kao što su povezivanje sa bazom podataka, rukovanje HTTP zahtevima ili rad sa sistemima datoteka.

Node.js se obično koristi za pravljenje brzih, skalabilnih Web aplikacija i popularan je u razvoju aplikacija u realnom vremenu kao što su aplikacije za ćaskanje, onlajn igre i platforme za video strimovanje uživo.

4.3.2.1 Notifications Service API

Notifications Service pruža API od par metoda pomoću kojih se mogu dodati i listati push subscription-i i poslati prilagođene push poruke. Za kreiranje API-ja je iskorišćena popularna biblioteka Express biblioteka. To je minimalistički Web framework koji pojednostavljuje proces kreiranja i održavanja Web servera, rukovanje HTTP zahtevima i rutiranje, kao i renderovanje prikaza. Pomoću Ekpress-a mogu se kreirati rute koje rukovode dolaznim HTTP zahtevima, postaviti middleware softver za obavljanje određenih radnji pre ili nakon obrade zahteva i koristiti mehanizme šablona za prikazivanje prikaza i vraćanje HTML-a ili drugih tipova odgovora klijentu. Takođe podržava nekoliko middleware biblioteka za rukovanje uobičajenim zadacima, kao što su parsiranje tela zahteva, rukovanje kolačićima i sesijama i serviranje statičkih datoteka.

```
1  const express = require('express');
2  const app = express();
3
4  app.get('/', (req, res) => {
5      res.send("Hello World!");
6  });
7
8  app.listen(3000, () => {
9      console.log("Server started on port 3000");
10 });
11
```

Slika 5. Primer jednostavnog Express API-ja

Primer jednostavnog Express API-ja koji sluša zahteve na portu 3000 i odgovara sa “Hello World” porukom na GET zahtev na ruti „/“ dat je na slici 5.

U ovom primeru prvo je uvežen Express module koristeći „require('express')“ i kreirana aplikacija korišćenjem „express()“ funkcije. Nakon toga je definisana ruta na putanji „/“ korišćenjem „app.get()“ metode i prosleđena je callback funkcija koja kao parametre uzima objekte zahteva i

odgovora. Callback funkcija se izvršava svaki put kada stigne GET zahtev na ovoj putanji. U ovom slučaju callback funkcija šalje „Hello World!“ string kao odgovor. Na kraju server se startuje na portu 3000 pozivom metode „app.listen()“ kojoj se kao parametri prosleđuju broj porta i callback funkcija koja se izvršava jednom kada se server startuje.

API metode Notifications servisa:

- POST metoda „/save-subscription“: Metoda čuva push subscription objekate u „PushNotificationSubscriptions“ table unutar baze servera. Kao query parametar prosleđuje se Id korisnika u bazi a kao telo zahteva push subscription objekat.
- GET metoda „/get-subscriptions“: Lista push subscription-e sačuvane u bazi. Kao query parametre prihvata offset i limit parametre koji se mogu koristiti za paginaciju.
- POST metoda „/send-notification-to-user“: Koristi se za slanje push poruke jednom korisniku. Kao query parametar se prosleđuje Id korisnika a objekat push poruke u telu zahteva.
- POST metoda „/broadcast-notification“: Emituje push poruku ka svim korisnicima, sadržaj push poruke se prosleđuje u telu zahteva.

4.3.2.2 Upravljanje push porukama

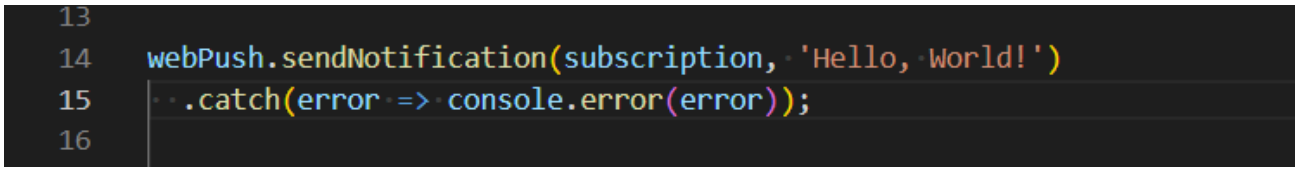
Za upravljanje push porukama iskorišćena je „Web-push“ biblioteka, koja se instalira korišćenjem Node Package Manager-a. Omogućava interakciju sa Web Push API-jem i slanje push poruka korisnicima. Biblioteka se uključuje u aplikaciju pozivom „require('Web-push')“ metode.

Zatim se podese javni i privatni VAPID ključevi (slika 6). VAPID je mehanizam koji omogućava aplikacionom serveru da se identifikuje na push servisu. Sadrži javni i privatni ključ, koji se koristi za potpisivanje JWT tokena koji se šalje sa push porukom.

```
2
3   const vapidKeys = {
4     publicKey: 'YOUR_PUBLIC_KEY',
5     privateKey: 'YOUR_PRIVATE_KEY'
6   };
7
8   webPush.setVapidDetails(
9     'mailto:example@yourdomain.org',
10    vapidKeys.publicKey,
11    vapidKeys.privateKey
12  );
13
```

Slika 6. Podešavanje VAPID ključeva

Push poruke se šalju pozivom „WebPush.sendNotification()“ metode (slika 7). Metoda kao parametre ima push subscription objekat i telo poruke koja se šalje. Push subscription objekat se kreira na strani klijenta tj u Service Worker-u pozivanjem „PushManager.subscribe()“ metode i čuva se u bazu pozivom „/save-subscription“ metode ovog servisa.



Slika 7. Primer slanja push poruke

Push poruke se od aplikacionog servera do notifickaciong servisa dopremaju koristeći RabbittMQ.

RabbitMQ je open-source softver za razmenu poruka koji implementira Advanced Message Queuing Protocol (AMQP). Glavna svrha posrednika poruka kao što je RabbitMQ je da omogući različitim delovima sistema da međusobno komuniciraju na slabo povezan način, slanjem i primanjem poruka. Ovo omogućava veću fleksibilnost, skalabilnost i toleranciju grešaka u distribuiranom sistemu.

RabbitMQ podržava više paterna za razmenu poruka kao što su publish/subscribe, point-to-point i request/response. Takođe pruža funkcionalnosti poput postojanosti poruka, potvrde isporuka i ugrađeni alati za upravljanje i praćenje. Može se koristiti u nizu različitih konteksta, kao što je implementacija arhitekture mikroservisa, sistema vođenih događajima i distribuiranih sistema. Takođe se često koristi u kombinaciji sa drugim tehnologijama kao što su Docker i Kubernetes.

Za razmenu poruka iskorišćen je publish/subscribe (kod RabbitMQ-a nazvan producer/consumer) mehanizam korišćenjem redova, gde je aplikacioni server producer (proizvođač) a notifikacioni servis consumer (potrošač) poruka. Red je bafer koji čuva poruke dok ih consumer ne potroši. Svaki red je identifikovan jedinstvenim imenom i može da sadrži više poruka. Kada se poruka pošalje u red, ona se postavlja na kraj reda i isporučuje se sledećem dostupnom potrošaču. Exchange (razmenjivač) je agent za rutiranje koji prima poruke od producer-a i gura ih u jedan ili više redova na osnovu skupa pravila, poznatih kao vezivanja (bindings). Svaka razmena je identifikovana jedinstvenim imenom i tipom, koji određuje algoritam rutiranja koji se koristi.

Postoje četiri tipa exchange-a u RabbitMQ-u:

- Direct exchange: usmerava poruke u redove na osnovu tačnog podudaranja ključa za rutiranje.
- Fanout exchange: usmeravanje poruka ka svim redovima koji su vezani za razmenu.
- Topic exchange: usmerava poruke u redove na osnovu podudaranja šablona ključa za rutiranje.
- Headers exchange: rutira poruke na osnovu vrednosti zaglavlja, a ne ključa za rutiranje.

Kada se poruka pošalje na razmenu, razmena koristi ključ za rutiranje i vezivanja da bi odredila u koje redove treba da bude isporučena poruka. Poruke se zatim stavljaju u odgovarajuće redove za potrošnju od strane potrošača.

Aplikacioni server i notifikacioni servis poruke razmenjuju koristeći Direct exchange nazvan „push-notifications-exchange“ i istoimenog reda.

Aplikacioni server se povezuje na Rabbit koristeći „RabbitMQ.Client“ biblioteku dostupnu kroz NuGet package manager. Kada se server pokrene izvrši se povezivanje i deklaracija reda i razmene ukoliko one već ne postoje, kroz „PushNotificationsBroker“ singleton klasu. Ona takođe omogućava slanje poruka na definisani red koristeći „PublishMessage“ metodu (primer poziva na slici 8).

Kada se na serveru obradi zahtev iz kojeg proizilazi push poruka, direktno iz kontrolera se poziva

„PublishMessage“ metoda kojoj se kao paramter prosleđuje objekat „BrokerMessage“ klase koja ima četiri svojstva:

- Title: string koji predstavlja naslov poruke.
- Message: string sadržaj poruke.
- SendToAll: boolean tip koji označava da li se poruka emituje ka svim korisnicima ili se šalje određenom korisniku.
- UserId: ukoliko je SendToAll postavljen na false UserId se prosleđuje kao identifikator korisnika kome se šalje push poruka.

```
49      [HttpPost("{pollId}")]
50      [ProducesResponseType(200)]
51      [ProducesResponseType(400)]
        0 references
52      public async Task<IActionResult> AddAnswers([FromRoute] long pollId, [FromBody] AddAnswersDto model)
53      {
54          if (!ModelState.IsValid)
55              return BadRequest();
56
57          try
58          {
59              var username = _httpContextAccessor.HttpContext.User.FindFirst(ClaimTypes.Name)?.Value;
60
61              var user = await _userAuthenticationRepository.GetUserByName(username);
62
63              var ownerId = await _answersRepository.AddAnswers(user.Id, pollId, model);
64
65              _pushNotificationsService.PublishMessage(new BrokerMessage()
66              {
67                  Title = $"New answer!",
68                  Message = "Someone just answered your poll.",
69                  SendToAll = false,
70                  UserId = ownerId,
71              });
72
73              return Ok();
74          }
75          catch (Exception e)
76          {
77              Log.Error(e, e.Message);
78
79              return BadRequest(e.Message);
80          }
81
82      }
83  }
84
```

Slika 8. Primer slanja obaveštеanja korisniku na čiju je anketu upravo dodat odgovor

Notifikacioni servis se pri pokretanju povezuje na RabbitMQ koristeći „amqplib/callback_api“ biblioteku, dostupnu kroz Node package manager. Nakon što se servis poveže na kanal za komunikaciju, definišu se i započinje konzumiranje poruka na dva exchange-a i reda ukoliko već ne postoje (slika 9). Prvi je, već pomenuti „push-notifications-exchange“, za komunikaciju sa aplikacionim serverom, a drugi je „buffer-exchange“ koji služi za baferovanje poruka koje se emituju ka svim korisnicima.

```
262         channel.bindQueue('push-notifications-exchange', 'push-notifications-exchange', '');
263         channel.consume('push-notifications-exchange', function (msg) {
264             try
265             {
266                 if (msg.content) {
267                     console.log("CONSUMED MESSAGE FROM API:", msg.content.toString());
268                     let data = JSON.parse(msg.content.toString());
269                     if (data.SendToAll) {
270                         addMessageToBroadcastQueue(data);
271                     }
272                     else {
273                         pushMessageToUser(data);
274                     }
275                 }
276             }
277             catch (e) {
278                 console.log(e);
279             }
```

Slika 9. Konzumiranje poruka za slanje ili emitovanje

Kada servis konzumira poruku on prvo proveri da li je namenjena jednom korisniku ili za emitovanje ka svim korisnicima.

Ukoliko je namenjena jednom korisniku ona se šalje direktno pozivom „pushMessageToUser“ metode. Metoda pribavi push subscription objekat iz baze na osnovu „UserId“ property-ja iz objekta primljene poruke, a nakon toga šalje poruku pozivom „Webpush.sendNotification“ metode (slika 10).

```
83  const pushMessageToUser = (data) => {
84      getSubscriptionByUserId(data.UserId, function(err, subscription) {
85          if (err !== null) {
86              console.error('UNABLE TO SEND NOTIFICATION TO: ', data.UserId)
87          }
88          else {
89              try {
90                  webpush.sendNotification(subscription, JSON.stringify({Message: data.Message, Title: data.Title}))
91                  .then(() => {
92                      console.log('NOTIFICATION SENT TO USER: ', data.UserId);
93                  })
94                  .catch((e) => {
95                      console.log(`NOTIFICATION IS NOT SENT TO USER: ${data.UserId}, error: ${e}`);
96                  });
97              }
98              catch (e) {
99                  console.log(e);
100             }
101         }
102     });
103 }
```

Slika 10. Slanje push obaveštenja određenom korisniku

Ukoliko je poruka namenjena za emitovanje ka svim korisnicima, ona se dodaje na red za emitovanje pozivom „addMessageToBroadcastQueue“ (slika 11). Objekat poruke koja se dodaje na red sadrži objekat primljene push poruke, ukupan broj korisnika kojima se šalje i pomeraj tj broj korisnika kojima je poruka već poslata (inicijalno 0).

```
71  const addMessageToBroadcastQueue = async (message) => {
72      getTotalNumberOfSubscriptions(function(err, total) {
73          if (err !== null) {
74              console.error('UNABLE TO GET SUBSCRITPIONS COUNT. MESSAGE WONT BE BROADCASTED.')
75              return;
76          }
77          console.log('PUBLISHING BROADCAST MESSAGE TO QUEUE, TOTAL SUBSCRIPTIONS: ', total)
78          publicChannel.sendToQueue('buffer-exchange',
79              Buffer.from(JSON.stringify({total: total, offset: 0, message: message})));
80      });
81  }
```

Slika 11. Dodavanje poruke za emtovanje na red za obradu

Kada se poruka konzumira sa „buffer-exchange“ reda poziva se „broadcastMessage“ metoda (slika 12) koja prvo pribavi push subscription objekte iz baze sortirane po integer identifikatoru, preskačući broj definisan pomerajem u poruci i uzimajući broj objekata dedinisan u konfiguraciji servera: „NUMBER_OF_SUBSCRIPTIONS_PER_BROADCAST“. Kada se pribave push subscription-i svakome se šalje poruka pozivom „Webpush.sendNotification“ metode. Nakon slanja poruka, pomeraj se uvećava za broj poslatih poruka i poruka se vraća nazad na red za ponovno konzumiranje.

Na ovaj način se izbegava zagušenje kada u sistemu postoji veliki broj korisnika jer se poruka emituje po etapama. Pošto rad za slanje poruka trajno čuva poruke sve dok ne budu konzumirane, ukoliko se notifikacioni servis zaustavi dok još uvek postoji poruka koja se emituje, kada se on pokrene ponovo emitovanje će biti nastavljeno.

```
39  const broadcastMessage = (notification) => {
40      try {
41          getSubscriptions(notification.offset, NUMBER_OF_SUBSCRIPTIONS_PER_BROADCAST, function(err, subscriptions) {
42              if (err !== null) {
43                  console.error('ERROR FETCHING ANY SUBSCRIPTIONS.');
44                  return;
45              }
46              if (subscriptions !== null) {
47                  subscriptions.forEach(subscription => {
48                      try {
49                          webpush.sendNotification(subscription, JSON.stringify(notification.message))
50                          .then(() => {
51                              console.log(`NOTIFICATION SENT TO: ${subscription.userId}`);
52                          })
53                          .catch((e) => {
54                              console.log(`'NOTIFICATION IS NOT SENT TO USER: ${subscription.userId}, error: ${e}`);
55                          });
56                      }
57                      catch(e) { console.log(e) }
58                  });
59              }
60              notification.offset += NUMBER_OF_SUBSCRIPTIONS_PER_BROADCAST;
61              if (notification.offset < notification.total) {
62                  publicChannel.sendToQueue('buffer-exchange', Buffer.from(JSON.stringify(notification)));
63              }
64          });
65      }
66      catch(e) {
67          console.error(e);
68      }
69  }
```

Slika 12. Emitovanje poruke ka više korisnika

4.3.3 Klijentska aplikacija (React PWA i Service Worker)

Klijentska aplikacija se sastoji od korisničkog interfejsa i Service Workera. Aplikacija je pisana korišćenjem React funkcionalnih komponenti i uz pomoć Material UI biblioteke, dok je za HTTP pozive za komunikaciju sa serverom korišćena Axios biblioteka. Na klijentskom delu je potrebno obezbediti mogućnost instaliranja aplikacije, zatražiti dozvolu od korisnika za prikazivanje push obaveštenja i registrovati Service Worker čiji je posao da kreira push subscription objekat, pošalje ga Notifications servisu i sluša „push" događaj za pristigle poruke.

4.3.3.1 React, funkcionalne komponente i hook-ovi

React je JavaScript biblioteka za izradu korisničkih interfejsa. React je open-source a razvio ga je, i održava ga Facebook. React koristi virtuelni DOM (Document Object Model) koji ga čini veoma efikasnim u ažuriranju korisničkog interfejsa kao odgovor na promene u stanju aplikacije. Kada se stanje komponente promeni, React će prvo ažurirati virtuelni DOM, a zatim će odrediti minimalni skup promena koji treba da se izvrši na realnom DOM-u, što rezultira efikasnijim i bržim procesom ažuriranja. React komponente su napisane u JavaScript-u i mogu se renderovati ili na serveru koristeći Node.js ili na klijentu pomoću pretraživača. React takođe podržava JSX, što je proširenje sintakse za JavaScript koje vam omogućava da pišete elemente slične HTML-u u JavaScript kodu.
U React-u funkcionalne komponente su JavaScript funkcije koje vraćaju React element. One su jednostavne, lake za razumevanje i mogu biti efikasnije od klasnih komponenti jer nemaju dodatne funkcionalnosti. Funkcijama se mogu proslediti parametri koji se mogu koristiti pri renderovanju. Na slici 13 se nalazi primer funkcionalne komponente.

```
1   function Example(props) {
2       return <h1>Hello, {props.name}</h1>;
3   }
```

Slika 13. Primer funkcionalne komponente

Funkcionalne komponente mogu koristiti hook-ove za upravljanje stanjem komponente. Hooks su funkcije koje omogućavaju da se „prikačite" na React funkcije, kao što su metode stanja i životnog ciklusa. Najprostiji i najkorišćeniji hook je „useState" (primer se može videti na slici 14), koji omogućava dodavanje stanja funkcionalnim komponentama. On uzima inicijalno stanje kao argument i vraća niz od dva elementa: trenutnu vrednost stanja i funkciju za ažuriranju stanja.

```
1   import { useState } from 'react';
2
3   function Counter() {
4     const [count, setCount] = useState(0);
5     return (
6       <div>
7         <p>You clicked {count} times</p>
8         <button onClick={() => setCount(count + 1)}>
9           Click me
10        </button>
11      </div>
12    );
13  }
```

Slika 14. Primer funkcionalne komponente sa „useState“ hook-om

Pored „useState“ hook-a korišćen je i „useEffect“ hook (slika 15) koji omogućava pokretanje sporednih efekata, kao što je preuzimanje podataka ili ažuriranje DOM-a, kao odgovor na promene stanja. Ima dva argumenta: callback funkciju i niz zavisnosti koji određuje koje vrednosti komponenti treba da trigeruju efekat.

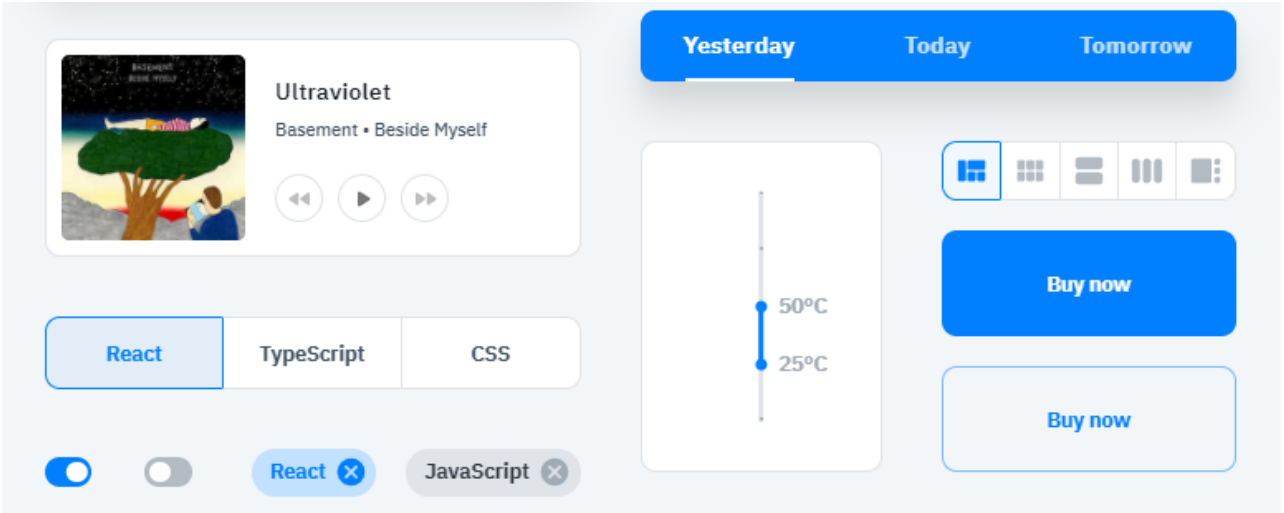
```
1    useEffect(() => {
2        // side-effect logic
3    }, [dependency1, dependency2, ...])
```

Slika 25. Primer „useEffect“ hook-a

4.3.3.2 Material UI

Material UI je popularna open-source biblioteka za izradu korisničkih interfejsa za Web, mobilne i desktop aplikacije. Implementira Material Design smernice razvijene od strane Google-a. Obezbeđuje skup unapred izgrađenih komponenti koje prate vizuelni jezik Material Design-a, kao što su dugmad, forme, polja za unos, komponente rasporeda i elementi za navigaciju. Ove komponente su dizajnirane da se koriste na konzistentan način u aplikaciji i potpuno su prilagodljive.

Material UI takođe pruža moćan sistem za tematiyaciju koji omogućava lako prilagođavanje izgleda po potrebama aplikacije. Mogu se menjati boje, tipografiju i druge vizuelne elemente. Primer izgleda komponenti se može videti na slici 16.



Slika 16. Primer MUI komponenti

React Material-UI takođe pruža responsive dizajn, što znači da se komponente automatski prilagođavaju veličini ekrana uređaja koji se koristi, omogućavajući kreiranje aplikacija koje izgledaju sjajno na bilo kojoj veličini ili orijentaciji ekrana. Primer korišćenja Material UI dugmeta u React komponenti se može videti na slici 17.

```
import * as React from 'react';
import Button from '@mui/material/Button';

export default function MyApp() {
  return (
    <div>
      <Button variant="contained">Hello World</Button>
    </div>
  );
}
```

slika 17. primer korišćenja Material UI dugmeta

4.3.3.2 Axios

Axios HTTP Client je popularna JavaScript open-source biblioteka koja omogućava HTTP pozive iz Web pretraživača i Node.js aplikacija. Široko se koristi za upućivanje API poziva i rukovanje rezultujućim podacima, kao što je JSON. Axios je zasnovan na promise-ima, što znači da vraća promise sa podacima kada je zahtev uspešan.

Neke od glavnih karakteristika Axios-a uključuju:

- Podršku za sve uobičajene HTTP metode (GET, POST, PUT, DELETE, itd.)
- Automatsku transformaciju podataka zahteva i odgovora (npr. JSON)
- Podrška za Promse API
- Presretanje zahteva i odgovora
- Prekidanje zahteva
- Podrški za XSRF (Cross-site request forgery) zaštitu

```
import axios from 'axios';

const { API_BASE_URL ='https://www.pocket-polls.click/api' } = process.env

export const request = async (url, method, data) => {
    return await axios({
        url: `${API_BASE_URL}/api/${url}`,
        method: method,
        data: data,
        timeout: 5000,
        headers: {
            'Content-Type': 'application/json-patch+json',
            'Authorization': 'Bearer ' + localStorage.getItem('user-token')
        },
    });
};
```

Slika 18. primer korišćenja Axios-a u aplikaciji

Na slici 18. se može videti primer korišćenja Axios-a u aplikaciji. Funkciji „request“ se kao parametri prosledjuju URL (endpoint na API-ju), HTTP metod i podaci koji su sadržaj tela zahteva. Kao osnovni URL je postavljen URL servera aplikacije i na njega se nadovezuju endpoint-i.

Pošto aplikacija koristi JWT tokene za autentifikaciju, kada se korisnik prijavi access token se sačuva u lokalno skladište pretraživača i kasnije se uz svaki zahtev pridružuje kao „Authorization“ header.

4.3.3.3 Dodavanje opcije instaliranja React Web aplikaciji

Da bi omogućili opciju instaliranja React aplikacije neophodno je odraditi sledeće:

1. Kreirati „manifest.json“ datoteku koja sadrži informacije o aplikaciji kao što su ime aplikacije, ikonice različitih veličina kao i URL sa koga se pokreće aplikacija. File je potrebno smestiti u *public* direktorijumu aplikacije.

2. Linkovati manifest datotoeku unutar „index.html“ stranice unutar „head“ sekcije (primer na slici 19).

3. Registrovati Service Worker.

4. Dodati osuškivanje na „beforeinstallpromt“ događaj. Ovaj događaj se okida kada korisnik poseti PWA, i on se može iskoristiti da se prikaže modul koji pita korisnika da li želi da instalira aplikaciju.

```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
5      <meta charset="utf-8" />
```

Slika 19. Primer linkovanja manifest.json fajla

Manifest datoteka tipično sadrži sledeće inforamcije (primer na slici 20):

- „short_name“: Kratko ime aplikacije koje se prikazuje na početnom ekranu ili u meniju sa aplikacijama.
- „name“: Puno ime aplikacije.
- „icons“: Niz ikonica koje se koriste za aplikaciju.
- „start_url“: URL sa kojeg se pokrece aplikacija kada se pokrene. Ulazna tačka u aplikaciju.
- „display“: Mod prikaza aplikacije koji može biti „standalone“, „fullscreen“ ili „minimal-ui“.
- „theme_color“: Boja teme za aplikaciju.
- „background_color“: Boja pozadine aplikacije.

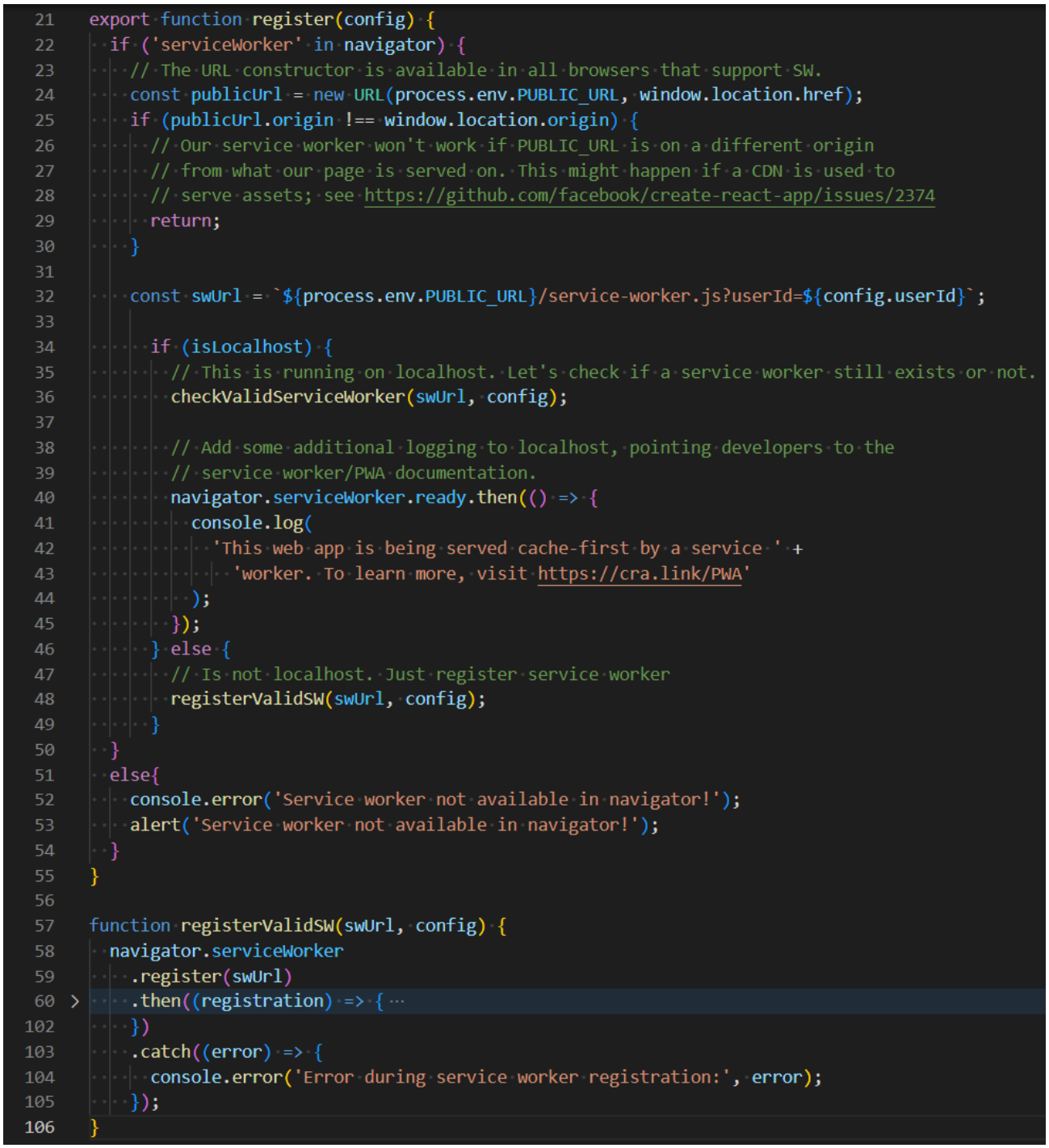
```
{
  "short_name": "polls",
  "name": "Pocket polls",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    },
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}
```

Slika 20. Primer manifest.json fajla

4.3.3.4 Registracija Service Worker-a

Web aplikacija zahteva od pretraživača da registruje Service Worker pozivanjem „navigator.serviceWorker.register()“ metode (dostupne u većini pretraživača), kojoj prosleđuje putanju do skripte Service Worker-a kao parametar. Pretraživač proverava da li je Service Worker skripta locirana na istom origin-u kao i Web aplikacija, ako nije registracija je neuspešna. Ako jesu, pretraživač preuzima sa servera skriptu Service Worker-a i instalira je tako što je smešta u keš. Pretraživač proverava da li već postoji aktivan Service Worker za aplikaciju. Ako već postoji, novi Service Worker ulazi u stanje čekanja sve dok se ne zatvore sve kartice pretraživača koje koriste već postojeći Service Worker. Kada se sve kartice zatvore, Service Worker postaje aktivni Service Worker stranice.

Kod koji izvršava registraciju nalazi se u „serviceWorkerRegistration.js“ datoteci. Registracija se vrši pozivom metode „register“ (slika 21). Poziv ove metode se uobičajeno izvršava unutar „index.js“ datoteke kod React aplikacija. Metoda proverava da li je „serviceWorker“ dostupan u „navigator“ objektu i ako jeste poziva „navigator.serviceWorker.register()“ metodu kojoj prosleđuje URL do datoteke Service Worker-a. Na URL se kao query parametar dodaje identifikator korisnika koji se kasnije koristi unutar Service Worker-a pri kreiranju push subscription objekta.



```
export function register(config) {
  if ('serviceWorker' in navigator) {
    // The URL constructor is available in all browsers that support SW.
    const publicUrl = new URL(process.env.PUBLIC_URL, window.location.href);
    if (publicUrl.origin !== window.location.origin) {
      // Our service worker won't work if PUBLIC_URL is on a different origin
      // from what our page is served on. This might happen if a CDN is used to
      // serve assets; see https://github.com/facebook/create-react-app/issues/2374
      return;
    }

    const swUrl = `${process.env.PUBLIC_URL}/service-worker.js?userId=${config.userId}`;

      if (isLocalhost) {
        // This is running on localhost. Let's check if a service worker still exists or not.
        checkValidServiceWorker(swUrl, config);

        // Add some additional logging to localhost, pointing developers to the
        // service worker/PWA documentation.
        navigator.serviceWorker.ready.then(() => {
          console.log(
            'This web app is being served cache-first by a service ' +
              'worker. To learn more, visit https://cra.link/PWA'
          );
        });
      } else {
        // Is not localhost. Just register service worker
        registerValidSW(swUrl, config);
      }
  }
  else{
    console.error('Service worker not available in navigator!');
    alert('Service worker not available in navigator!');
  }
}

function registerValidSW(swUrl, config) {
  navigator.serviceWorker
    .register(swUrl)
    .then((registration) => { ···
    })
    .catch((error) => {
      console.error('Error during service worker registration:', error);
    });
}
```

Slika 21. Metoda za registrovanje Service Worker-a

U ovoj datoteci se, takođe, nalazi i „requestNotificationsPermission“ metoda (slika 22) koja se poziva da bi korisniku prikazali modal kojim se zahteva dozvola za prikazivanje push obaveštenja. Metoda poziva „requestPermission“ metodu Notification API-ja. Ona je sastavni deo pretraživača i omogućava prikaz obaveštenja van Web stranice, čak i kada je Web aplikacija zatvorena. Važno je imati na umu da korisnik mora da dozvoli Web aplikaciji da prikazuje obaveštenja pre nego što obaveštenja mogu da budu prikazana. Takođe, upit za dozvolu biće prikazan samo na bezbednom poreklu (HTTPS), osim ako se sajt ne doda na početni ekran. U ovom slučaju može da se prikaže na lokalnom hostu ili drugom lokalnom nebezbednom origin-u. Važno je napomenuti da Notification API nije podržan od strane svih Web pretraživa i uređaja, a proces prikazivanja obaveštenja može da varira između različitih platformi.

```
export const requestNotificationPermission = async () => {
    const permission = await window.Notification.requestPermission();
    // value of permission can be 'granted', 'default', 'denied'
    // granted: user has accepted the request
    // default: user has dismissed the notification permission popup by clicking on x
    // denied: user has denied the request.
    if(permission === 'denied'){
        throw new Error('Permission not granted for Notification!');
    }
    else if (permission === 'default') {
        setTimeout(async () => await requestNotificationPermission(), 3000)
    }
    else {
        console.log("User has granted permission for push notifications.")
    }
}
```

Slika 22. Metoda kojom se zahteva dozvola za prikazivanje obaveštenja

U slučaju naše aplikacije, bitan je trenutak u kome se registruje Service Worker, jer nam je potreban identifikator korisnika u bazi aplikacionog servera kako bi njemu pridružili push subscription objekat koji kreira Service Worker pri aktivaciji. Zbog toga se registracija poziva tek nakon što se korisnik registruje i prijavi na aplikaciji (slika 23). Istovremeno se zatraži i dozvola od korisnika.

```
        await http.request('users/login', 'POST', {
                userName: data.username,
                password: data.password
            }).then(response => {
                if (response.status === 200 && response.data.token !== null)
                {
                    localStorage.setItem('user-token', response.data.token);

                    serviceWorkerRegistration.register({
                        userId: response.data.userId,
                        onUpdate: function(registration) {
                            console.log('Service worker registred successfully');
                        },
                        onSuccess: function(registration) {
                            console.log('Service worker updated successfully');
                        }
                    });

                    serviceWorkerRegistration.requestNotificationPermission();
```

Slika 23. Prijavljivanje korisnika i traženje dozvole za prikaz obaveštenja

4.3.3.5 Sadržaj Service Worker-a

Posao Service Worker-a je da pri registraciji kreira push subscription objekat i sačuva ga na Notifications servisu i da sluša „push“ događaj kako bi prihvatio pristigla obaveštenja i prikazao ih pozivom Notifications API-ja.

Za pretplaćivanje na push obaveštenja iskorišćen je „activate“ događaj (slika 24). Push subscription objekat se kreira pozivom „self.registration.pushManager.subscribe()“ metode, kojoj se kao parametar prosleđuju opcije koje uključuju javni VAPID ključ i opcije prikaza.

Nakon toga se push subscription objekat zajedno sa identifikatorom korisnika, koji se uzima iz query parametara URL-a sa kojeg je učitan Service Worker, šalju notifikacionom servisu pozivom *„/save-subscription“* metode. Ukoliko je push subscription već sačuvan za korisnika on će biti zamenjen novim.

Kačenjem na „push“ događaj (slika 25) servie worker osluškuje push poruke koje emituje Notification servis preko Push Servisa. Kada se primi poruka parsira se naslov i sadržaj poruke i prikazije pozivom „showNotification“ metode koja se nalazu unutar „registration“ objekta worker-a. U pozadini se poziva Notification API pretraživača koji poziva sistemske metode za prikaz modala.

Kao dodatne opcije push obaveštenja moguće je dodati ikonicu koja se prikazuje unutar modala push obaveštenja, i opciju koja podešava modal da zavibrira kada se pojavi.

```
100  const createOrUpdatePushSubscription = async () => {
101    try {
102      const applicationServerKey = urlB64ToUint8Array(WEB_PUSH_PUBLIC_KEY)
103      const options = { applicationServerKey, userVisibleOnly: true }
104      const subscription = await self.registration.pushManager.subscribe(options)
105
106      let userId = new URL(location).searchParams.get('userId');
107      const SERVER_URL = `${NOTIFICATIONS_API_URL}/save-subscription?userId=${userId}`;
108
109      var response = await fetch(SERVER_URL, {
110        method: 'post',
111        headers: {
112          'Content-Type': 'application/json',
113        },
114        body: JSON.stringify(subscription),
115      })
116
117      if (response.status === 201) {
118        console.info('SUBSCRIBED TO PUSH NOTIFICATIONS SERVER!')
119      }
120      else {
121        console.err('FAILED TO SUBSCRIBE TO PUSH NOTIFICATIONS SERVER!')
122      }
123
124    } catch (err) {
125      console.error('Error', err)
126    }
127  }
128
129  self.addEventListener('activate', async () => {
130    // This will be called only once when the service worker is installed for first time.
131    await createOrUpdatePushSubscription()
132  })
133
134  self.addEventListener('pushsubscriptionchange ', async () => {
135    await createOrUpdatePushSubscription()
136  })
```

Slika 24. Kreiranje i čivanje push subscription objekta

```
self.addEventListener('push', function(event) {
    if (event.data) {
        console.log('Push event: ', event.data.text())
        let data = JSON.parse(event.data.text());
        showLocalNotification(data.Title, data.Message, self.registration)
    }
})

const showLocalNotification = (title, body, swRegistration) => {
    const options = {
        body,
        icon: `${process.env.PUBLIC_URL}/logo192.png`,
        vibrate: true
    }
    swRegistration.showNotification(title, options)
}
```

Slika 25. Dodavanje osluškivanja na „push“ događaj

4.3.4 Nginx API Gateway

Kako bi obezbedili serviranje svih delova sa aplikacije sa istog origin-a i preko HTTPS protokola neophodan je registrovani domen i validan SSL sertifikat.

SSL (Secure Sockets Layer) sertifikat, poznat i kao digitalni sertifikat, je datoteka koja sadrži informacije o identitetu Web lokacije i organizaciji koja ga poseduje, kao i autoritetu za sertifikate (certificate authority - CA) koji ga je izdao. SSL sertifikati se koriste za uspostavljanje enkriptovane veze između Web servera i Web pretraživača, obezbeđujući da svi podaci koji se razmenjuju između njih budu poverljivi i bezbedni. Kada korisnik poseti Web lokaciju sa SSL sertifikatom, Web pretraživač će prikazati ikonu katanca na traci za adresu i URL Web lokacije će početi sa „https“ umesto „http“, što ukazuje da je veza bezbedna.

Nginx je Web server i reverse proxy server. Poznat je po visokim performansama i maloj upotrebi resursa, što ga čini popularnim izborom za Web aplikacije sa velikim prometom. Može koristiti za opsluživanje statičkog i dinamičkog sadržaja, kao i za upravljanje SSL/TLS enkripcijom i HTTP/2 vezama. Takođe ima ugrađene mogućnosti load balansiranja i keširanja, omogućavajući mu da distribuira dolazni saobraćaj na više servera i poboljša performanse Web lokacije. Pored toga, Nginx se često koristi kao reverse proxy, što znači da može primati dolazni saobraćaj i prosleđivati ga različitim Web serverima na osnovu URL-a ili drugih kriterijuma. Nginx se široko koristi i dostupan je za mnoge operativne sisteme uključujući Linux, Windows, i macOS.

Nginx je iskorišćen kao reverse proxy postavljen ispred ostalih aplikacija servera. On je jedina aplikacija kojoj je moguće pristupiti od spolja. Nginx svakom pristiglom zahtevu na portu 443 (podrazumevani HTTPS port) pridružuje SSL sertifikat u odgovoru, i preusmerava zahtev na osnovu rute ka drugim aplikacijama.

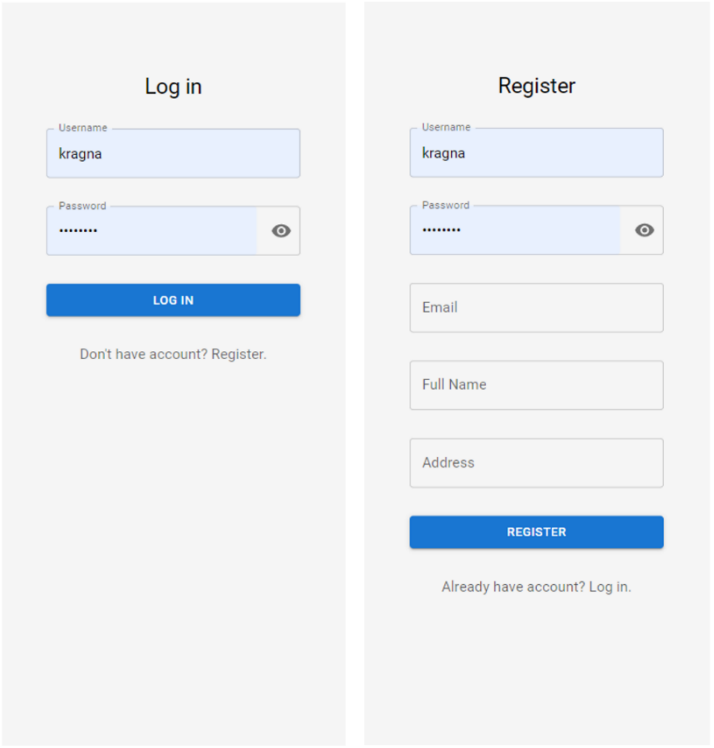
Sva pravila Nginx servera se dodaje u „nginx.conf“ datoteku (slika 26). Svi zahtevi koji u ruti sadrže „/api/“ se preusmeravaju na aplikacioni server, tj na .NET Web API koji se podrazumevano pokreće na 5000 portu. Sve zahteve koji u ruti sadrže „/push-notifications/“ se preusmeravaju na API Notifications servisa. A svi preostali zahtevi se preusmeravaju na port 3000 gde se nalazi sadržaj klijentskog dela aplikacije.

```
server {
        listen 443 ssl http2;
        server_name pocket-polls.click www.pocket-polls.click;

        server_tokens off;
        ssl_certificate /etc/letsencrypt/live/pocket-polls.click/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/pocket-polls.click/privkey.pem;

        location / {
                rewrite ^ https://$host$request_uri? permanent;
        }

        location /api/ {
                rewrite ^/api/?(.*)$ /$1 break;
                proxy_pass http://polls-api:5000;
        }

        location /push-notifications/ {
                rewrite ^/push-notifications/?(.*)$ /$1 break;
                proxy_pass http://push-notifications-api:4000;
        }

        location ~ /.well-known/acme-challenge {
                allow all;
                root /var/www/html;
        }
}
```

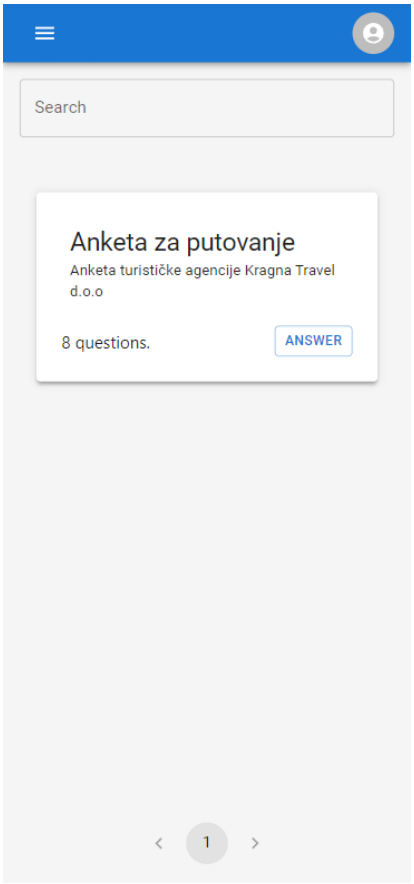
Slika 26. Sadržaj nginx.conf fajla

4.4 Primer rada aplikacije

Sledi primer rada aplikacije koji će biti opisan počevši od same registracije. Korisnik koji nije ranije prijavljen tj kada prvi put poseti Web aplikaciju, početna tačka je „SignIn“ stranica (slika 27), gde korisnik može da kreira nalog ili da se prijavi ukoliko već poseduje nalog. Ako se korisnik registruje, popunjavanjem potrebnih polja i pritiskom na dugme „register“ se šalje zahtev ka aplikacionom serveru. Ukoliko je zahtev validan tj korisničko ime ili email nisu zauzeti i lozinka je validna, korisnik će automatski biti prijavljen i preusmeren na „Home“ stranicu. Ukoliko zahtev za registraciju ili prijavljivanje nije validan prikazaće se odgovarajuća validaciona poruka unutar popup prozora.



Slika 27. Izgled stranice za prijavu i registraciju



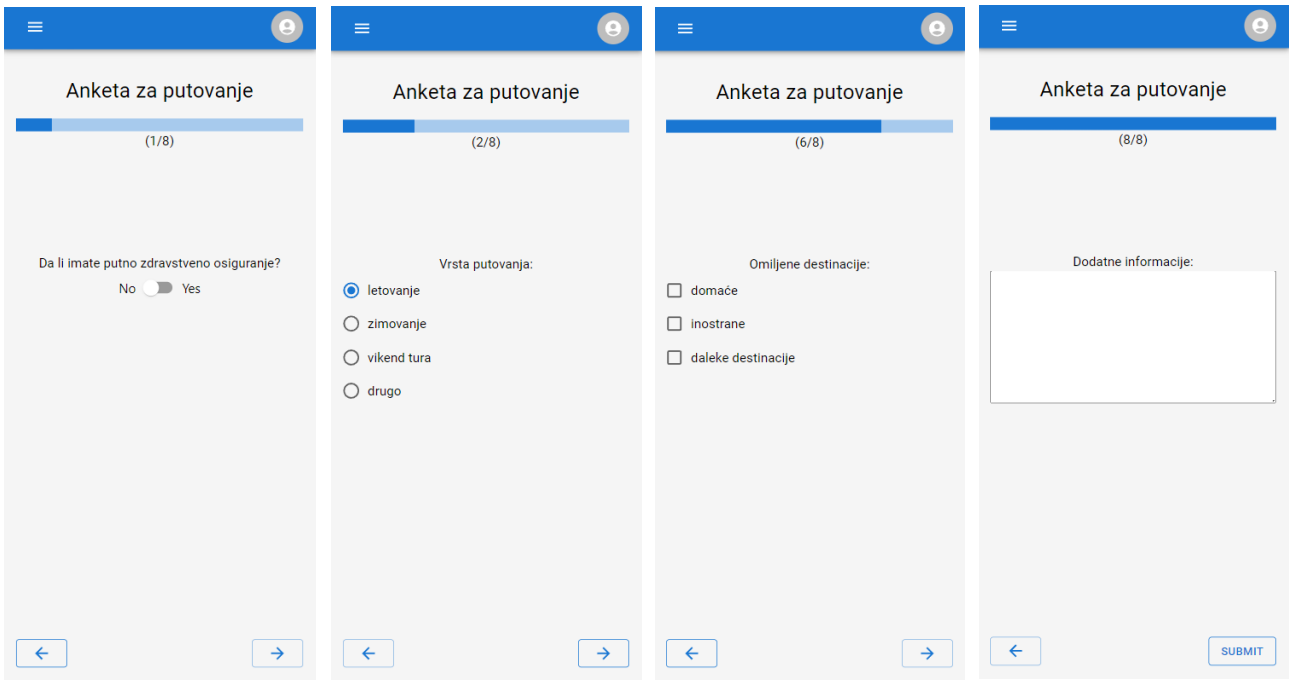
Slika 28. Početna stranica

Ako je korisnik već registrovan i prijavljen na aplikaciju početna tačka je „Home“ tj početna stranica (slika 28). Na početnoj stranici su izlistane sve kreirane ankete, od svih korisnika. Mogu se pretraživati po imenu i opisu popunjavanjem polja „Search“.

Ukoliko broj kartica koje predstavljaju ankete premašuje veličinu ekrana, moguće je navigirati se pomoću paginacije na dnu ekrana.

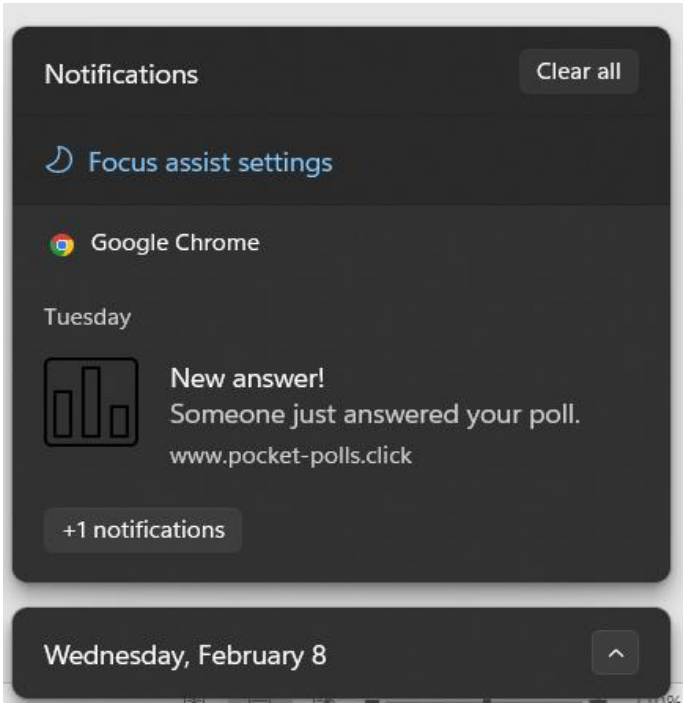
Pritiskom na dugme “answer” unutar kartice koja predstavlja anketu, otvara se nova stranica “Poll” tj sama anketa i počinje popunjavanje.

Na slici 29. je prikazana stranica za popunjavanje anketa. Svaki tip pitanja ima svoju komponentu, korisnik se kroz pitanja navigira korišćenjem dugmića na dnu ekrana, dok se na vrhu nalazi progress bar. Kada se dođe do poslednjeg pitanja pojavljuje se "Submit" dugme na čiji klik se popunjena anketa šalje serveru aplikacije. Ukoliko su svi odgovori validni, aplikacija navigira korisnika nazad na početni ekran,. Ukoliko nije, pojaviće se popup sa validacionom porukom. U trenutku kada je odgovor uspešno prosleđen serveru aplikacije, korisniku koji je vlasnik popunjene ankete se šalje push obaveštenje sa porukom da je upravo dobio novi odgovor.



Slika 29. Stanica za popunjavanje ankete

Kada korisnik primi push obaveštenje ono će biti prikazano u baru za obaveštenja operativnog sistema. Na slici 30 se može videti kako to igleda na Windows 11 operativnom sistemu.



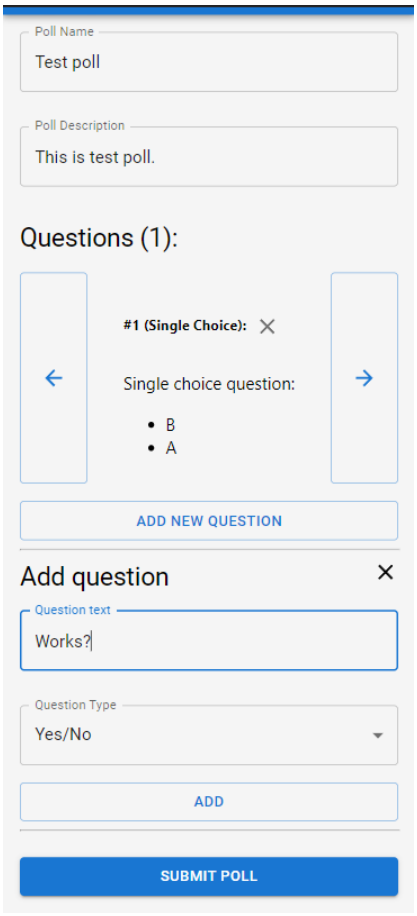
Slika 30. Primer prikazane push poruke na Windows-u

Na „Create Poll“ stranici (slika 31) se mogu kreirati ankete. Potrebno je uneti naziv i opis ankete i dodati makar jedno pitanje. Pitanja se dodaju u „Add question“ sekciji gde se unosi tekst pitanja i bira se jedan od četiri tipa pitanja:

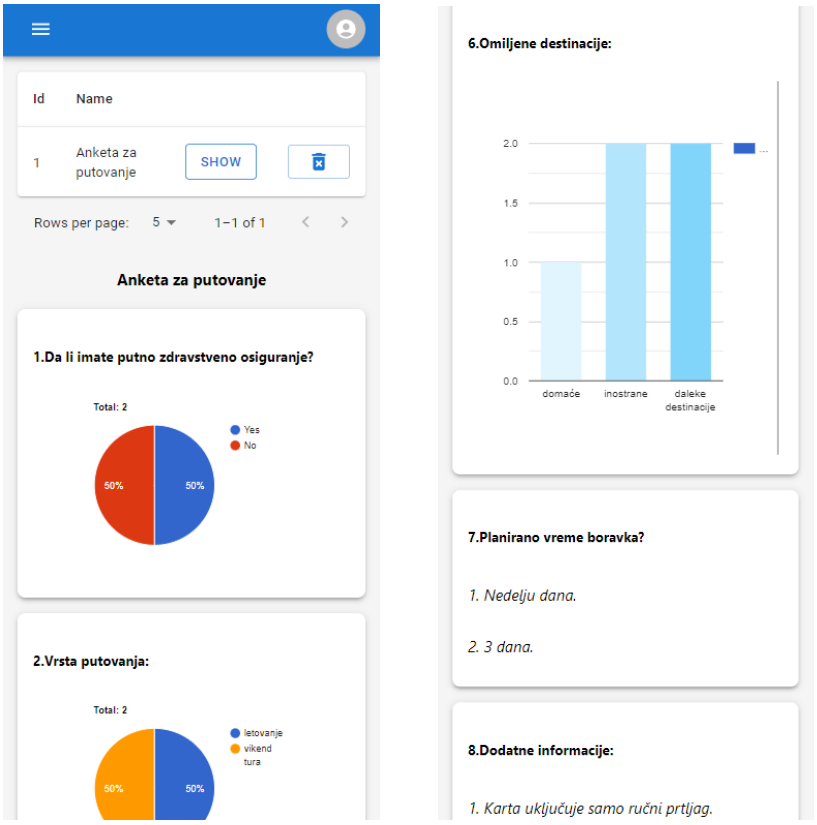
1. Yes/No
2. Single choice
3. Multiple choice
4. Text

Dodata pitanja se mogu prelistavati u sekciji iznad i moguće ih je izbaciti iz liste. Klikom na „Submit Poll“ dugme se anketa šalje serveru aplikacije. Ukoliko anketa nije validna biće prikazan popup prozor sa validacionom porukom, a ukoliko jeste u prozoru će pisati da je anketa uspešno kreirana. Istovremeno ka svim korisnicma će biti emitovano push obaveštenje sa porukom da je kreirana nova anketa na koju mogu dodati svoj odgovor.

Na „Answers“ stranici (slika 32) moguće je pregledati sve odgovore na anketu. Pitanja se prikazuju različito u zavisnosti od tipa pitanja. Yes/No i Single choice tipovi pitanja se prikazuju kroz pie chart-ove, Multiple choice tip se prikazuje kao column chart dok se Text tip pitanja izlistavuju unutar liste. Na ovoj stranici je takođe moguće obrisati anketu.



Slika 31. Strtanica za kreiranje ankete



Slika 32. Sranica za pregledanje odgovora anketa

5. ZAKLJUČAK

React Progressive Web Aplikacije su dobar izbor za pravljenje Web aplikacija kojima se može pristupati sa bilo kog uređaja, uključujući mobilne uređaje. PWA su dizajnirane da rade offline, da budu brze i responsivne i da pružaju iskustvo nalik native aplikacijama.

React PWA treba izabrati u odnosu na native Android i iOS aplikacije ako je cilj da se dopre do što više korisnika i podrži više platformi sa jednom bazom koda. Pored toga React PWA su obično brže i jeftinije za razvoj od native aplikacija. Međutim ako aplikacija zahteva pristup funkcijama specifičnim za uređaj koje nisu dostupne preko Web API-ja, native aplikacija može biti bolji izbor.

Web Push Protokol i Service Worker-i su moćni alati za kreiranje Web aplikacija koje u offline modu i pružaju bolje iskustvo korisnicima. Međutim, oni takođe imaju neka ograničenja:

- Podrška pretraživača: Web Push Protokol i Service Worker-e podržavaju samo moderni Web pretraživači, tako da Web aplikacije koje koriste ove funkcije možda neće raditi na starijim pretraživačima ili određenim uređajima.
- Bezbednost: Service Worker-i i Web Push Protokol zahtevaju bezbednu vezu (HTTPS) za proces registracije, tako je da bi se sprečili napadi i zaštitili podaci korisnika.
- Ograničeni scope: Service Worker-i imaju ograničen scope, mogu da kontrolišu samo stranice koje se učitavaju nakon završetka procesa registracije i na istom origin-u kao i worker.
- Ograničeni kapacitet skladištenja: Service Worker-i mogu da koriste Cache API za skladištenje asset-a, ali on ima ograničenje količine podataka koji se mogu uskladištiti. Ovo može biti ograničenje za Web aplikacije koje moraju da skladište veliku količinu podataka.
- PWA push obaveštenja: Push obaveštenja ne podržavaju svi Web pretraživači i uređaji, a čak i kada su podržana, proces njihovog prikazivanja može da varira između različitih platformi.
- Ograničena funkcionalnost: Service Worker-i su ograničeni na određene funkcije, na primer, ne mogu direktno da pristupe DOM-u i ne mogu da pristupe određenim API-jima pretraživača, kao što su pristup clipboard-u, kameri i još mnogo toga.
- Ograničena kontrola korisnika: Korisnici mogu samo da kontrolišu worker-a preko podešavanja pretraživača, a programer može da kontroliše servisera samo preko koda i ne postoji način da komunicira sa korisnikom.
- Ograničeno debagiranje: Otklanjanje grešaka može biti teško jer se pokreću u posebnoj niti i imaju ograničen pristup alatima za razvoj pretraživača
- Push obaveštenja nisu garantovana za isporuku, a takođe mogu biti odložena, u zavisnosti od uslova mreže i broja obaveštenja koje šalje server.

Treba napomenuti da Notification API nije podržan na Internet Exploreru. Pored toga, mogućnost prikazivanja push obaveštenja na mobilnim platformama varira. Na iOS-u, push obaveštenja se prikazuju pomoću Apple Push Notification usluge (APNS), koja zahteva dodatno podešavanje i konfiguraciju.

Implementacija push obaveštenja u React PWA može biti malo kompleksinija u odnosu na native aplikacije, zato što zahteva dodatna podešavanja i konfiguraciju.

Nasuprot PWA, kod native aplikacija, push obaveštenja su tipično integrisana direktno u operativni sistem, gde programeri mogu koristiti obezbeđene API-je za registrovanje i upravljanje push obaveštenjima.

Međutim, uz pomoć biblioteka kao što je „react-pwa-push“, implementacija može biti lakša, možete koristiti biblioteku za upravljanje registracijom Service Worker-a i prijavljivanjem/otkazivanjem push obaveštenja.

Može se zaključiti da iako je proces podešavanja push obaveštenja u React PWA možda nešto komplikovaniji u odnosu na native aplikacije, to je i dalje relativno jednostavan proces, a kada se postavi, trebalo bi da funkcioniše besprekorno.

6. LITERATURA

[1] Generic Event Delivery Using HTTP Push (RFC 8030), *https://datatracker.ietf.org/doc/html/rfc8030*

[2] Push API specifikacija, *https://www.w3.org/TR/push-api/*

[3] React.js dokumentacija, *https://reactjs.org/docs/getting-started.html*

[4] web.dev: Learn PWA, *https://web.dev/learn/pwa/*

[5] MDN Web docs: Service Worker API, *https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API*

[6] Express.js dokumentacija, *https://expressjs.com/en/api.html*

[7] Material UI dokumentacija, *https://mui.com/material-ui/getting-started/overview/*

[8] .NET dokumentacija, *https://learn.microsoft.com/en-us/dotnet/*

[9] Rabbit MQ dokumentacija, *https://www.rabbitmq.com/documentation.html*