SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 934

USPOREDBA JAVASCRIPTOVE KNJIŽNICE REACT I RADNOG OKVIRA NEXT.JS ZA RAZVOJ WEB APLIKACIJA

Tin Prvčić

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 934

USPOREDBA JAVASCRIPTOVE KNJIŽNICE REACT I RADNOG OKVIRA NEXT.JS ZA RAZVOJ WEB APLIKACIJA

Tin Prvčić

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 10. ožujka 2023.

ZAVRŠNI ZADATAK br. 934

Pristupnik:	**Tin Prvčić (0036535899)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	izv. prof. dr. sc. Alan Jović

Zadatak: **Usporedba JavaScriptove knjižnice React i radnog okvira Next.js za razvoj web aplikacija**

Opis zadatka:

JavaScript je jedan od temeljnih programskih jezika World Wide Weba i najvažniji jezik za razvoj na strani klijenta. Neka proširenja JavaScripta u vidu dodatnih knjižnica i radnih okvira osmišljena su da omogućuju i razvoj na poslužiteljskoj strani, primjerice Node.js i Next.js. U ovom radu potrebno je opisati te kvalitativno i kvantitativno usporediti knjižnicu React i radni okvir Next.js koji služi za izgradnju web aplikacija koje se renderiraju na poslužiteljskoj strani uz korištenje jezika JavaScript. U radu je potrebno usporediti koliko je korištenje Next.js-a umjesto Reacta ili dodatno uz React povoljno ili nepovoljno te u kojim situacijama bi bilo opravdano. U tu svrhu, potrebno je izraditi jednu ili više jednostavnih primjenskih web aplikacija u obje tehnologije koje treba ispitati kako bi se utvrdilo koja je tehnologija bolja za koju primjenu.

Rok za predaju rada: 9. lipnja 2023.

Sadržaj

Uvod

Moderan web pokreće JavaScript. U zadnjem desetljeću, u svijetu JavaScripta razvijene su mnoge nove tehnologije, knjižnice i radni okviri, koji su nam omogućile bolji i interaktivniji web, no nauštrb kompleksnosti i ponekad performansi.

React.js je jedna od mnogih, ali i najpopularnijih, JavaScriptovih knjižnica koja nam olakšava razvoj modernih web aplikacija i web stranice. Aplikacije i stranice pisane Reactom renderiranju se klijentski, što znači da funkcioniraju slično kao i mobilne aplikacije. Prilikom prvog posjeta web aplikaciji, preglednik preuzme sav potreban JavaScriptov kod, te ga pokreće lokalno, na korisnikovom računalu, odakle (tek nakon što se preuzme i učita sav JavaScript) i dohvaća sve podatke. Taj prvi dohvat nešto je sporiji, ali nakon toga aplikacija dohvaća samo podatke i znatno je brža od klasične web stranice.

Model je konceptualno jednostavan, efikasan i brz, kako performansama tako i za razvoj, ali s tim dolaze i neki nedostaci. Već spomenuti problem dužeg početnog učitavanja aplikacije uvećan je na slabijim uređajima (pametnim telefonima) gdje aplikacije vrlo lako postanu previše zahtjevne pa to prvo učitavanje traje predugo, a ne pomaže ni činjenica da se prvi podaci mogu dohvatiti tek nakon što se aplikacija učita na uređaju. Unatoč tome, najveći problem je što Google, Twitter i slični servisi takve web aplikacije ne znaju parsirati - oni čitaju samo gotov html zapis stranice, koji je prazan - pa ne mogu generirati pretpregled takvih aplikacija i stranice, što između ostalog rezultira i time da ih internetske tražilice kažnjavaju stavljajući ih niže na popisu rezultata pretraživanja.

Next.js kao radni okvir oko React.js knjižnice pokušava riješiti te, ali i mnoge druge probleme. Next.js radi na principu renderiranja na poslužiteljskoj strani, što znači da se sav JavaScript prvo pokrene i evaluira na poslužitelju, a zatim se rezultat (zapis gotove stranice u html formatu) šalje na klijentsku stranu (web preglednik na korisnikovom računalu). Uz njega šalje se i JavaScriptov kod potreban da stranica postane interaktivna i nastavi raditi. Jasno, dohvaćaju se i prvi podaci koji služe za renderiranje stranice na poslužiteljskoj strani, ali se predaju i JavaScriptu, pa se taj prvi dohvat na klijentskoj strani može preskočiti.

Iako se konceptualno čini kao idealno rješenje, Next.js je radni okvir koji sa sobom povlači priličan broj nedostataka, pa će se ovaj rad fokusirati na razlike između knjižnice React.js i

okvira Next.js i razmatrati u kojim slučajevima bi korištenje Nexta uz React bilo pogodnije nego korištenje isključivo Reacta, a kada bi sam React možda bio bolji odabir.

1. Metodologija

Za potrebe ovoga rada izrađene su dvije web aplikacije: jedna (gotovo potpuno) statična web aplikacija (tipičan *landing page* nekog proizvoda ili tvrtke) s dvije stranice; jedna je statična a druga uz statičan sadržaj dohvaća i nešto podataka. Druga aplikacija je primjer klasične *admin dashboard* aplikacije s jednom statičnom i dvije dinamične stranice, pri čemu je na jednoj moguće i dodavati i brisati podatke, izmjene koje se odmah trebaju vidjeti na toj stranici.

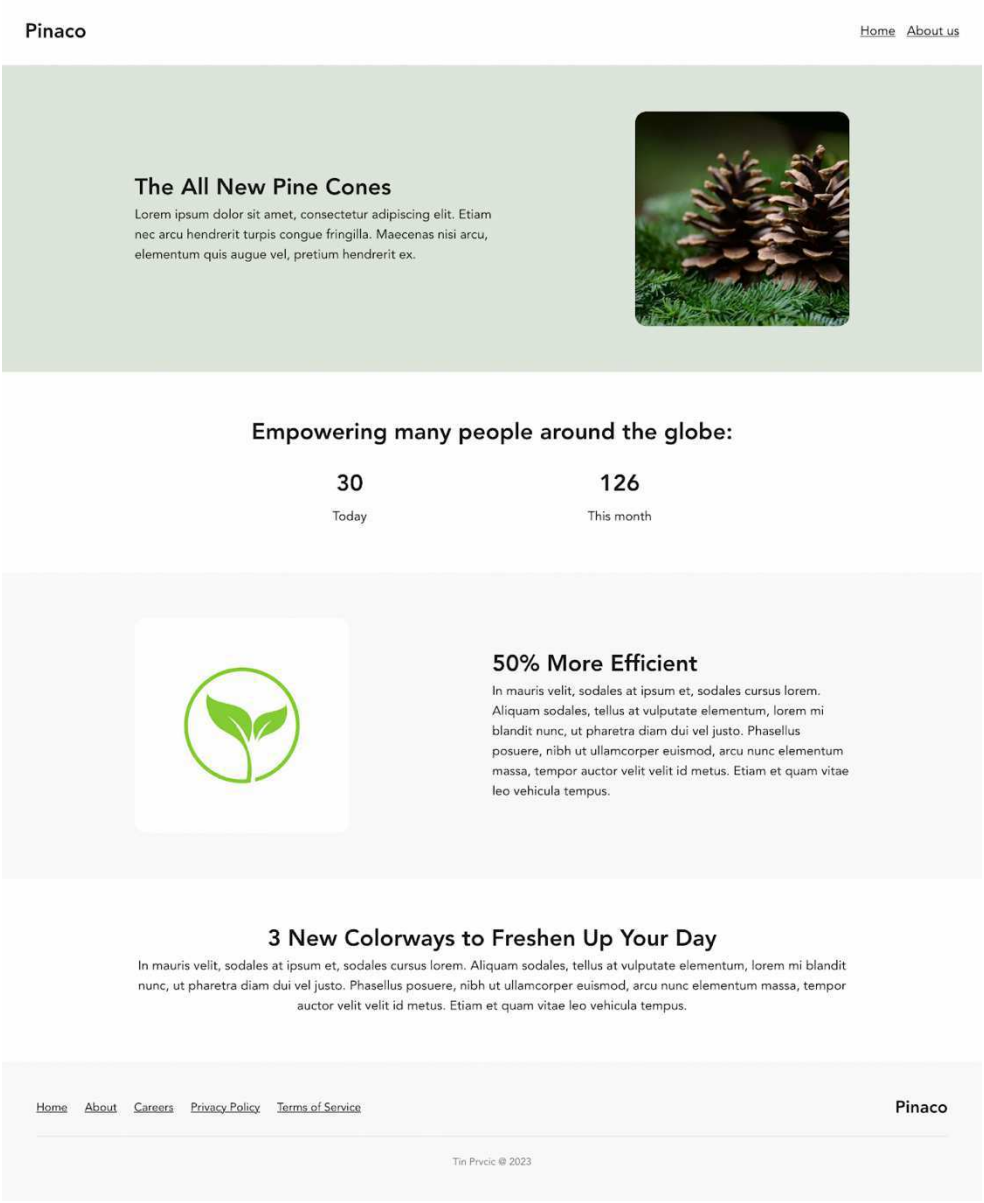
Svaka od aplikacija izrađena je u Reactu, a zatim uz potrebne izmjene adaptirana za Next.js, te optimirana prema najboljim praksama radnog okvira. Kao pomoć u svim aplikacijama korištena je mini-knjižnica "styled-components" koja pomaže u stiliziranju web aplikacije kroz JavaScript. Osim te knjižnice, statična aplikacija izrađena je isključivo koristeći React, bez korištenja ikakvih dodatnih knjižnica (koje nisu kritične za funkcioniranje Reacta), a za dinamičnu *dashboard* aplikaciju neke od značajki koje pruža Next.js popunjene su drugim knjižnicama. Na taj način možemo usporediti React.js i Next.js u "sterilnoj okolini", ali i staviti ih u neku stvarniju situaciju, s kojom bismo se možda susreli i u industriji.

Next.js donosi svoj *bundler* (alat koji pretvara kod pogodan za razvoj u kod koji se može pokrenuti u web pregledniku), a za React aplikacije korišten je "Vite", minimalan alat koji u posljednje vrijeme glasi i kao najpopularniji.

U sklopu rada analizirat ćemo razne parametre samog razvoja aplikacija u svakoj od tehnologija poput jednostavnosti razvoja, pomoći koju nam radni okvir/knjižnica pruža ili ne pruža i raznih optimizacija koje se događaju bez uplitanja razvojnog programera, ali ćemo se osvrnuti i na parametre krajnje razvijenih aplikacija: veličinu završnog JavaScriptovog paketa, vrijeme proteklo do trenutka kada se aplikacija učita i bude spremna za korištenje, kao i cjelokupnu potrošnju podataka i još neke parametre.

2. Statična web aplikacija

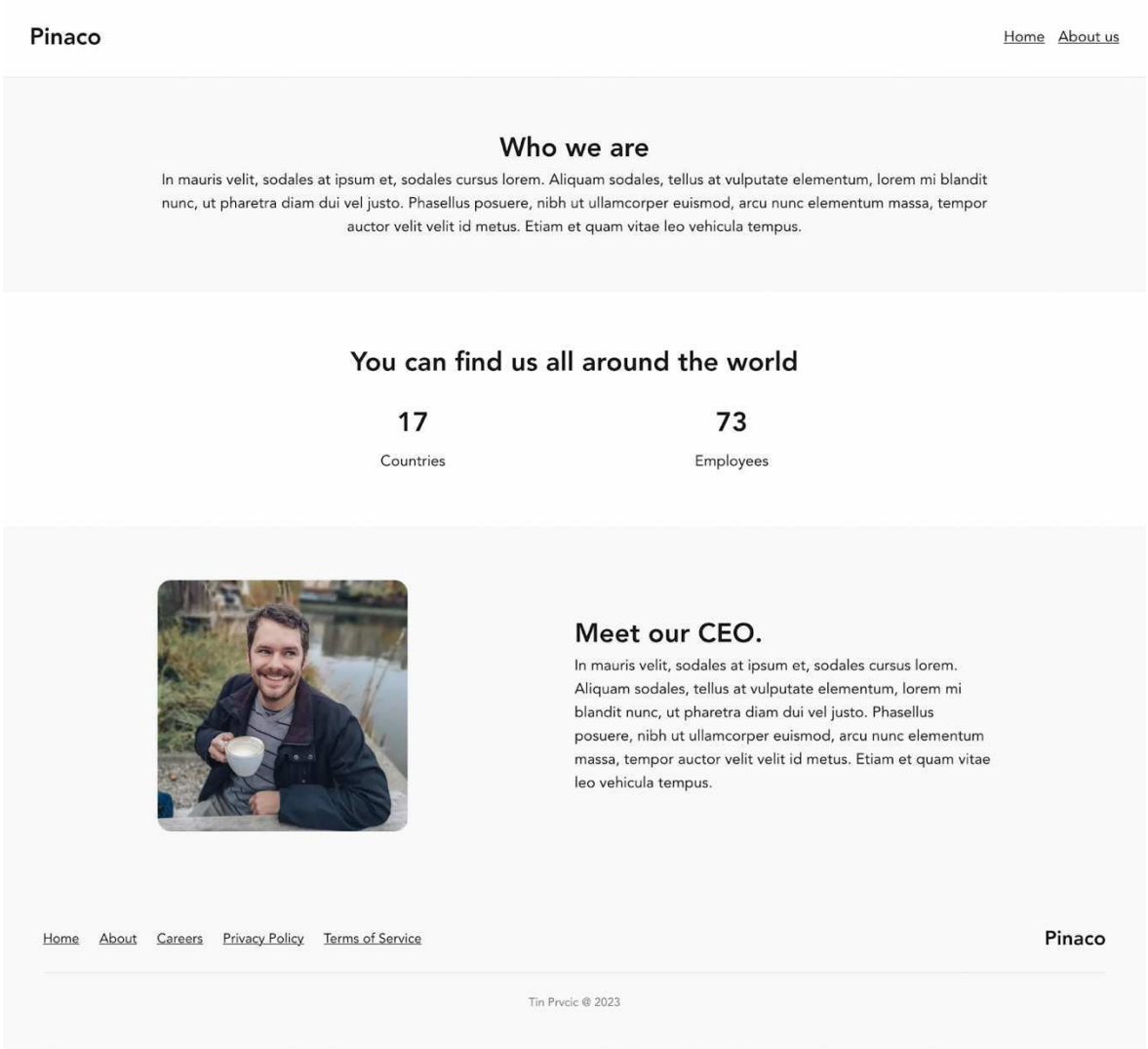
U ovom poglavlju usporedit ćemo React.js i Next.js prilikom razvoja statičnih web aplikacija. Naša statična web aplikacija sastoji se od dvije stranice. Prva stranica je klasična *landing page*, tj. naslovna stranica (Slika 2.1), na kojoj se većinom nalaze statični podaci (zapisani u kodu) uz nekoliko slika, a uz to se dohvaćaju i dvije brojke (vidljive na Slici 1. iznad teksta "Today" i "This month") koje se prikazuju *above the fold*, tj. pri vrhu su stranice, pa će na računalu će vidljive bez potrebe za pomicanjem stranice prema dolje.



Slika 2.1 Početna stranica statične web aplikacije

Brojke se u čistoj React verziji aplikacije dohvaćaju s klijentske strane, te se prije nego se učitaju prikazuje zamjenski tekst (--), dok se u Next.js verziji aplikacije dohvaćaju prilikom renderiranja na serveru pa nije potrebno prikazivati taj zamjenski tekst.

Druga stranica je potpuno statična, tj. nema nikakvog dohvata podataka već je sav potreban sadržaj zapisan u kodu. To je stranica “O nama” (Slika 2.2), a na njoj se opet nalazi jedna fotografija.



Slika 2.2 Stranica "O nama" statične web aplikacije

Sve ostale poveznice u podnožju stranice vode na nepostojeću stranicu (404), za koju ćemo također vidjeti da ju Next.js implementira nešto drugačije (bolje) nego to možemo postići čistim Reactom.

Aplikacija je napravljena na način da bude potpuno prilagodljiva veličini zaslona, tj. da funkcionira i na malim i na velikim uređajima (što je malo zakompliciralo razvoj prilikom

korištenja radnog okvira Next.js, kako ćemo vidjeti nešto kasnije). Fotografije korištene u aplikaciji preuzete su s usluge Pixabay [1].

2.1. Razvoj aplikacije

Aplikacija je prvo izrađena koristeći samo React, uz dodatak knjižnice styled-components radi lakšeg stiliziranja same aplikacije. Projekt je kreiran koristeći već spomenuti Vite, uz opciju korištenja jezika TypeScript koji proširuje JavaScript na način da omogućava tipiranje podataka (omogućava nam da pridružimo tip podataka varijabli ili konstanti – broj, logički tip, znakovni niz ili neki kompleksniji tip). Vite je generirao osnovnu strukturu projekta: datoteke potrebne za upravitelja paketima npm, datoteku `.gitignore`, konfiguracijske datoteke za jezik TypeScript, linter eslint i sam Vite. Uz to, stvorio je i demo mini-aplikaciju od jedne stranice koja služi kao dobar kostur za projekt. Osim toga, paket styled-components radi bez ikakve dodatne konfiguracije.

Za razliku od Reacta, Next.js sa sobom donosi više strukture i sam inicijalizira git repozitorij s prvim, inicijalnim *commitom*, što je korisno, pogotovo za programere početnike. Next.js nam predodređuje strukturu, pa tako svaka stranica mora biti smještena u svoj poddirektorij u direktoriju `pages`, dok je dostupan i direktorij `api` u kojem bismo pisali sučelja na poslužiteljskoj strani (taj dio Nexta nećemo razmotriti jer se bavimo klijentskom stranom). Next dolazi i sa svojom uslugom za optimiranje slika, koju bismo inače morali održavati zasebno, ili pak platiti neku uslugu koja to radi za nas.

U slučaju Reacta, većinu značajki potrebno je bilo implementirati od početka. Tako, na primjer, dohvat podataka potrebno je ostvariti ručno, tj. svaki puta kada bismo koristili komponentu koja konzumira neke podatke, potrebno ih je ručno dohvatiti, i pritom pratiti stanje dohvaćanja (učitavanje, uspjeh - podaci, greška), eventualno omogućiti ponovni dohvat podataka ako se desi greška, i prema tome prikazivati ispravno korisničko sučelje. U Nextu dovoljno je uz samu stranicu koja se prikazuje definirati i dohvat podataka, koje Next sam predaje aplikaciji na klijentu, a mi ne moramo razmišljati o stanju učitavanja jer ono ne postoji. Ako se dogodi greška prilikom dohvata podataka, potrebno ju je samo proslijediti Nextu (`throw` u JavaScriptu, a ako koristimo najnoviju sintaksu `async/await`, nije potrebno raditi ni to).

Navigacija je druga razlika, koja je možda i najveća. U Reactu je potrebno implementirati svoju navigaciju, jer ju sam React ne pruža. Tako je potrebno dvosmjerno sinkronizirati

unutarnje stanje aplikacije s adresom koju pokazuje web preglednik te direktno upravljati API-jevima preglednika kako bismo mijenjali, dodavali i slušali na promjene adrese. S obzirom na to stanje, potrebno je prikazati ispravnu stranicu, ili neku generičku stranicu ako ne postoji ništa na zadanoj adresi (`404`). Next.js donosi svoju komponentu `Router` koja umjesto nas brine za sve detalje navigacije. Stranice su organizirane u već spomenutom direktoriju `pages` i njegovim poddirektorijima, pa ne moramo na jednom mjestu definirati kada ćemo prikazivati koju stranicu, a i nekome tko ne radi na projektu lakše je razumjeti strukturu poddirektorija nego jednu veliku datoteku sa svim definicijama, pogotovo na većim projektima.

Zapravo bismo većinu prednosti Nexta (u pogledu razvoja aplikacije i tzv. *developer experience*) mogli pripisati tome što je to radni okvir, a ne samo knjižnica, pa nam donosi puno više značajki koje bismo sami trebali implementirati, ali projektu daje i nekakvu strukturu, što je vrlo vrijedno, pogotovo većim projektima, timovima i tvrtkama.

Međutim, moramo se dotaknuti i nedostataka prilikom unošenja Nexta u projekt. U ovoj manjoj i jednostavnijoj aplikaciji nije ih bilo mnogo. Glavni problem nije direktno problem Nexta već je rezultat renderiranja na poslužitelju. Uzmimo za primjer naslovni tekst na stranici, koji je centriran na manjim uređajima (pametnim telefonima), a poravnat lijevo na većim uređajima. Na React-način bismo to ostvarili *hookom* koji bi nam vraćao veličinu zaslona kategoriziranu u neku od kategorija. *Hook* bi prilikom inicijalizacije pročitao širinu zaslona iz objekta `window` u pregledniku, a zatim slušao na daljnje promjene veličine zaslona (Kôd 2.1)

```
export const useBreakpoints = (): {
  md: boolean;
  lg: boolean;
  current: "sm" | "md" | "lg";
} => {
  const [isMd, setMd] = useState(window.innerWidth >=
Breakpoints.md);
  const [isLg, setLg] = useState(window.innerWidth >=
Breakpoints.lg);

  useEffect(() => {
    const action = () => {
      setMd(window.innerWidth >= Breakpoints.md);
      setLg(window.innerWidth >= Breakpoints.lg);
    };

    window.addEventListener("resize", action);
    return () => {
      window.removeEventListener("resize", action);
    };
  }, []);

  return { md: isMd, lg: isLg, current: isMd ? "md" : isLg ?
"lg" : "sm" };
};
```

Kôd 2.1 – Hook za čitanje grupirane veličine zaslona

Zatim bismo kroz JavaScript konfigurirali Reactovu komponentu (kod nije priložen radi jednostavnosti, ali pretpostavimo da prima atribut `center`) tako da centrira tekst.

```
const ConditionallyCenteredParagraph = () => {
const { md } = useBreakpoints();
return <Heading center={!md}>Uvjetno centriran naslov<Heading
/>
}
```

No, taj pristup ne funkcionira u Nextu (ili općenito renderiranju na poslužitelju), iz razloga što objekt `window` nije dostupan na poslužitelju (poslužitelj nikako ne može znati dimenzije zaslona korisnikovog uređaja, kao ni druge informacije sadržane u objektu `window`), pa moramo pristupiti problemu na druge načine. Najčešći način jest da u projekt uvedemo pomoćne CSS-ove klase koje bi istu stvar radile pomoću *CSS media querya*, ali u

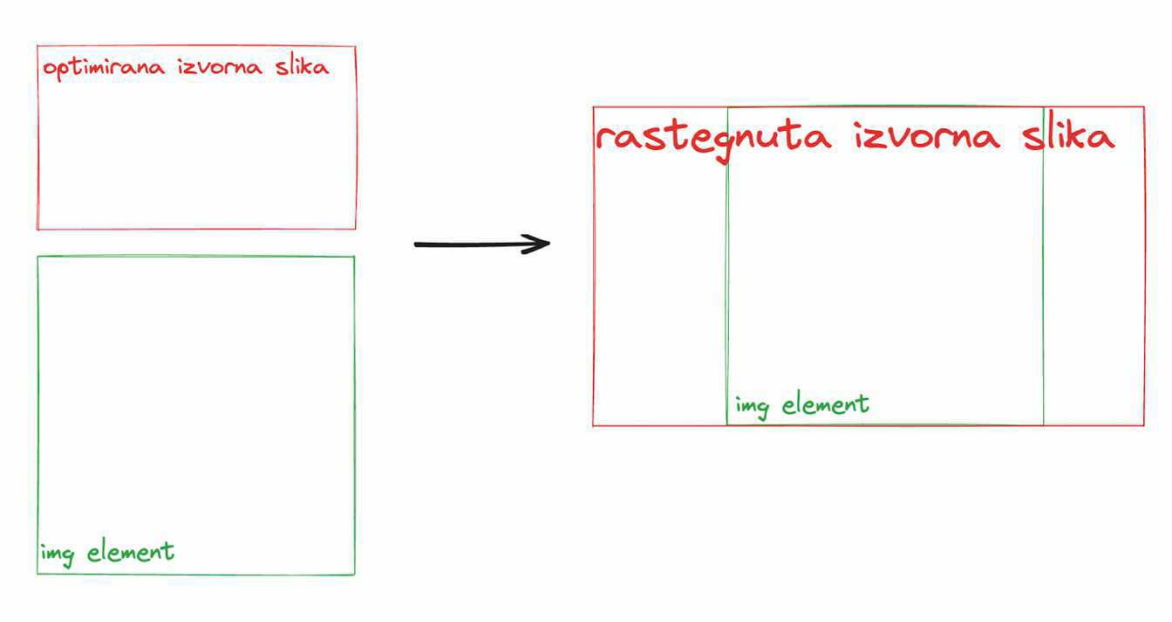
ovom slučaju, s obzirom na to da koristimo knjižnicu styled-components, prirodniji način bio je stvoriti stiliziranu verziju komponente `Heading` koja bi obavljala isti posao, samo na manje standardan (za React) način.

```
const StyledHeading = styled(Heading)`
  text-align: center;
  @media (min-width: ${Breakpoints.md}px) {
      text-align: start;
  }
`;
```

U ovom slučaju to ograničenje ne unosi previše dodatne kompleksnosti u projekt, no u većim projektima može imati više utjecaja. Također, razmotrimo sljedeći problem: kad bismo trebali prikazati određen broj natuknica poredanih vodoravno, jednu kraj druge (na primjer oznake u e-pošti ili slično), na način da ne prelaze u novi red ali dolaze do kraja trenutnog (Slika 2.3), to ne bismo nikako mogli ostvariti na poslužitelju, već bismo trebali pričekati da se kod učita na klijentskoj strani pa tamo izračunati broj natuknica koje je potrebno prikazati.

Slika 2.3 Prikaz problema s oznakama

Još jedan nedostatak koji se manifestira u ovoj aplikaciji jest već spomenuta optimizacija slika. Naime, prilikom optimizacije veličine Next u obzir uzima samo širinu prikazane slike pa, u slučaju kad je izvorna slika znatno šira nego prikaz slike (i da se izvorna slika rasteže preko samog elementa `img`, dakle `object-fit: cover`), slika će biti razvučena na dimenziju veću od dimenzije optimirane izvorne slike, pa će biti mutna (Slika 2.4).



Slika 2.4 Prikaz rastezanja slike koju Next.js optimizira

Ukratko možemo zaključiti da, kao i svi radni okviri, Next.js pokriva mnogo slučajeva, ali ih ne može sve pokriti savršeno, pa ćemo ako trebamo dodatnu kontrolu posegnuti ka rješenjima po mjeri. Za većinu korisnika Next.js odrađuje sasvim dovoljan posao s opcijama koje nudi i na ovakvoj vrsti aplikacije znatno olakšava razvoj, s obzirom na to da standardizira razvoj aplikacije i dodaje korisne značajke koje bismo inače morali implementirati sami.

2.2. Krajnja aplikacija

U ovom poglavlju razmatramo pakete JavaScripta koji se šalju na korisnikovo računalo, kao i ukupnu količinu podataka koja se šalje na klijentsko računalo i sveukupno završenu aplikaciju.

Naime, Next.js, iako služi primarno za renderiranje Reactovih aplikacija na poslužitelju, donosi svoj *runtime*, tj. donosi dodatan kod u korisnikov web preglednik, ali to nije sasvim

negativno, već mu omogućava korisne stvari poput automatskog razdvajanja koda u više datoteka. Nakon što kompajliramo obje aplikacije za produkciju (kompajliranjem za produkciju *bundler* radi razne optimizacije koda kako bi smanjio njegovu krajnju veličinu, ali i povećao kompatibilnost sa starijim web preglednicima koji ne podržavaju sve značajke modernog JavaScripta ili sva programska sučelja koja podržavaju moderni web preglednici) usporedit ćemo dobivene pakete JavaScripta. Usporedimo prvo osnovne kosture aplikacija koje generiraju Vite i Next.js - bez ikakvih izmjena. Uspoređujemo stvarnu veličinu datoteka, ali i veličinu datoteka nakon što na njih primijenimo kompresiju gzip (Tablica 2.1).

Tablica 2.1 Veličine završnih JavaScriptovih paketa kostura aplikacije

	React.js	Next.js
gzip	46 kB	82 kB
stvarna veličina	143 kB	250 kB

Iz ovih podataka vidimo da je veličina početne Next aplikacije podosta veća - čak 70% veća u slučaju kad koristimo kompresiju gzip. No, kako razvijamo aplikaciju i dodajemo svoj kod, razlika (u postotku) će biti sve manja, a u većim aplikacijama ta razlika zna postati i zanemariva. Next.js nam pruža i automatsko razdvajanje koda po stranicama, nešto što React ne može ponuditi jer nije svjestan naših stranica. U Reactu je moguće ručno razdvojiti kod, no to nema smisla u ovakvoj aplikaciji jer bismo morali dodavati stanja učitavanja tamo gdje to nije potrebno (jer je aplikacija statična), pa ćemo tu opciju razmotriti u dinamičnoj aplikaciji. Uzevši sve to u obzir, veličina datoteka krajnje aplikacije navedena je u tablici (Tablica 2.2) - za Next.js prvo su navedene ukupne veličine datoteka nakon primjene kompresije gzip koje odgovaraju ukupnoj količini učitanih podataka na određenim stranicama (/home i /about), a zatim i ukupna veličina JavaScripta.

Tablica 2.2 Veličine završnih JavaScriptovih paketa statične aplikacije

	React.js	Next.js
/home - gzip	-	108 kB
/about - gzip	-	108 kB
ukupno - gzip	63 kB	118 kB
ukupno - stvarna veličina	194 kB	375 kB

Na ovoj veličini aplikacije razlika u veličini JavaScripta je ~85%, što je malo veće nego razlika veličine kostura aplikacija. To je slučaj jer smo koristili razne dodatke koje Next.js pruža (posebno `next/router` i `next/image`). No, veličina JavaScriptovih paketa ne priča cijelu priču, pa ćemo usporediti i ukupnu količinu podataka koji se učitaju na otvaranje početne stranice (/home), te ju razložiti na komponente - slike, html, JavaScript i podaci. Ovdje gledamo mjeru *transferred* u konzoli preglednika Web Developer (Tablica 2.3).

Tablica 2.3 Količina podataka prenesena na otvaranje početne stranice statične aplikacije

	React.js	Next.js
html	704 B	5,74 kB
slike	289,73 kB	40,44 kB
javascript	63,18 kB	112,33 kB
podaci	289 B	-
ukupno	353,2 kB	158,51 kB
ukupno - bez slika	63,47 kB	118,07 kB

Nextova ugrađena optimizacija slika uvelike je smanjila ukupnu količinu podataka koju korisnikov web preglednik mora učitati, no ako zanemarimo slike, i dalje Nextova aplikacija preuzima skoro dvostruko više podataka.

Preostaje nam još izmjeriti stvarnu brzinu učitavanja web stranice, što ćemo učiniti s Googleovim alatom Lighthouse. Lighthouse je alat unutar prozora Web Developer u Google Chromeu koji analizira (između ostalog) brzinu učitavanja web stranica po raznim parametrima, pri čemu primjenjuje ograničenje snage procesora te brzine interneta, i tako simulira prosječan mobilni uređaj, ili desktop računalo ako se za to odlučimo [2]. Testiranje je obavljeno lokalno, kako bismo izbjegli ikakve mogućnosti za razlikama u testovima. Razmotrit ćemo metrike koje nam daje Lighthouse (Tablica 2.4).

- *First Contentful Paint* (**FCP**) - Vrijeme proteklo od početka učitavanja stranice do pojavljivanja nekakvog sadržaja na zaslonu [3].
- *Largest Contentful Paint* (**LCP**) - Vrijeme proteklo od početka učitavanja stranice do trenutka kada se glavni sadržaj stranice učitao - glavni sadržaj definiran je kao najveći blok teksta ili slika koja je vidljiva na zaslonu (bez pomicanja stranice) [4].
- *Total Blocking Time* (**TBT**) - Količina vremena u kojem je glava dretva web stranice bila zauzeta obavljajući duge zadatke. To je bitna informacija jer nam govori o tome koliko će stranica biti interaktivna odmah nakon učitavanja [5].

Tablica 2.4 Rezultati statične aplikacije u Lighthouseu

	React.js	Next.js
FCP	1.1 s	0.6 s
LCP	2.6 s	2.4 s
TBT	40 ms	20 ms

Iako veća, aplikacija u Next.jsu je generalno brža. Najveću razliku čini *First Contentful Paint*, koji je značajno manji zbog toga što Next.js koristi renderiranje na poslužitelju. Poslužitelj, osim što ima više snage, ne mora preuzeti i učitati sav kod pa brže generira

početnu stranicu, koju odmah šalje korisniku. *Last Contentful Paint* je vrlo sličan kod obje aplikacije, no Next je opet nešto brži, vjerojatno jer radi optimizaciju slika pa se one nešto brže učitaju. Metrika *Total Blocking Time* je zanemarivo mala kod obje aplikacije. Tu možemo pretpostaviti da će, kod većih projekata, Next.js biti u prednosti radi svojeg automatskog dijeljenja koda - za svaku stranicu će se jednostavno morati preuzeti i učitati manje JavaScriptovog koda, što zahtijeva manje resursa CPU-a.

Velika je razlika i u načinu rada obje aplikacije. Dok kompajliranjem Reactove aplikacije dobivamo optimirani JavaScriptov kod koji možemo poslužiti na bilo kojem http poslužitelju (npr. apache2 ili nginx), Next.js koristi svoj poslužitelj koji renderira html iz JavaScripta, te poslužuje i JavaScript i html. Bitno je još spomenuti i rukovanje stranicama koje ne postoje. Next.js, budući da ima svoj poslužitelj, može i vratiti ispravan HTTP-ov statusni kod, 404. React to ne može učiniti, jer sam poslužitelj koji poslužuje JavaScriptove datoteke ne zna postoji li stranica kojoj se pristupa ili ne. Vraćanje ispravnog statusnog koda nije samo semantički ispravnije nego pomaže i prilikom rangiranja na internetskim tražilicama.

2.3. Zaključak

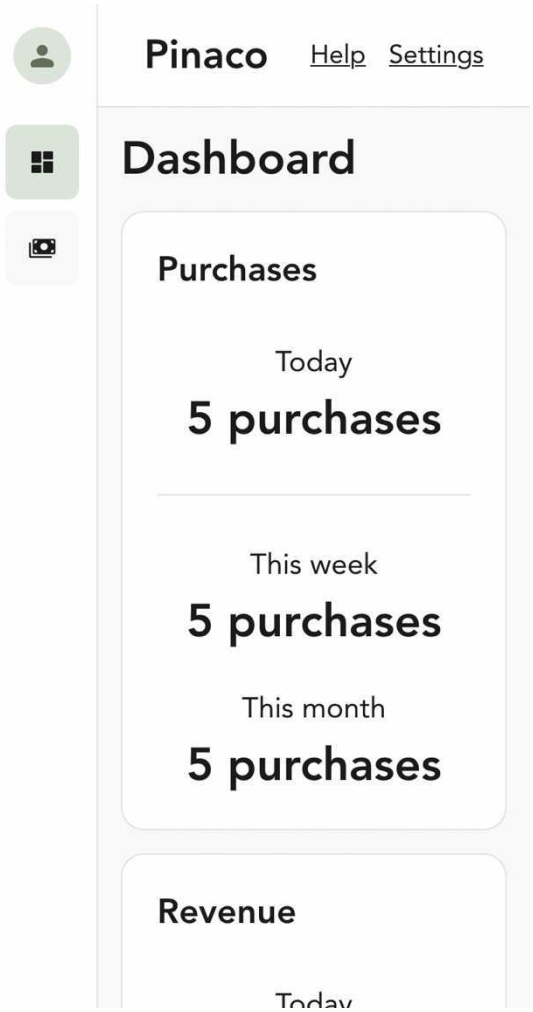
Razvoj ovakve aplikacije uz Next.js donosi određene probleme, ali Next sveukupno znatno olakšava razvoj aplikacije. Next.js projektu daje potrebnu strukturu, ali i značajke koje su jednostavne, a za većinu projekata ove naravi pokrivaju sve funkcionalnosti. Unatoč tome, glavni razlog za odabrati Next.js naspram Reacta je renderiranje na poslužitelju.

Za web stranice koje nisu skrivene iza prijave, uglavnom statične web stranice, često je bitno da budu dobro rangirane na Googleu i drugim internetskim pretraživačima. U takvim slučajevima renderiranje na klijentu jednostavno nije mogućnost, s obzirom na to da pretraživači ne znaju čitati takve web stranice pa ih penaliziraju. Next.js za takve projekte pruža odličan radni okvir nad Reactom, pa im omogućava vrlo lagano dodavanje interaktivnosti na web aplikaciju bez glavnih nedostataka renderiranja na klijentu.

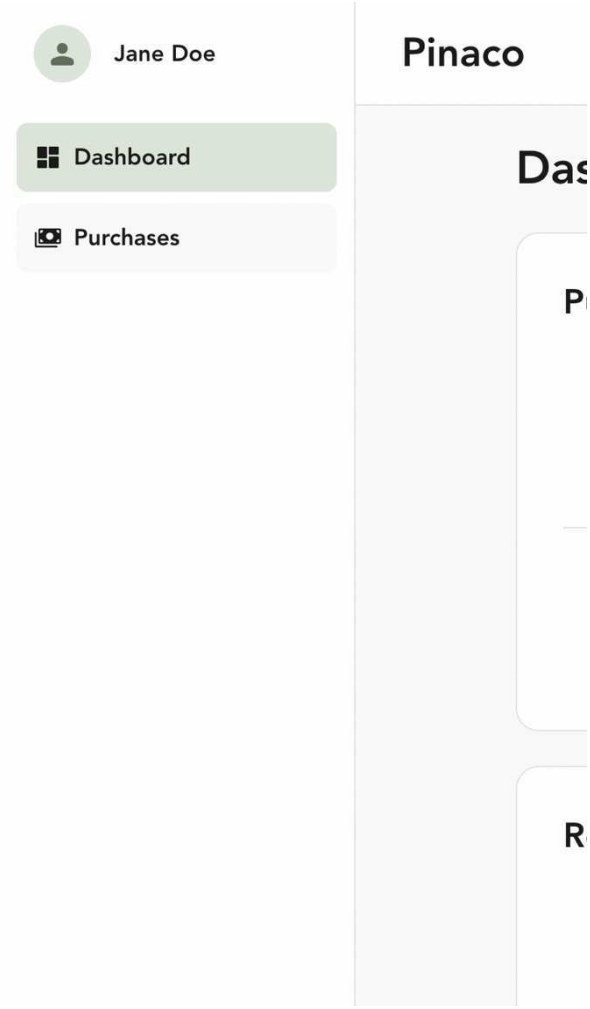
3. Dinamična web aplikacija

U ovom poglavlju usporedit ćemo korištenje React.js-a i Next.js-a za razvoj dinamičnih web aplikacija. Naša se dinamična web aplikacija sastoji od 3 stranice te oponaša tipičnu *admin dashboard* aplikaciju, tj. kontrolnu ploču za upravljanje nekim sustavom. Za potrebe ovog rada, aplikacija nema implementiranu autentifikaciju, ali možemo pretpostaviti da će u stvarnosti ovakav tip aplikacije biti sakriven i dostupan samo za autentificirane korisnike.

Na lijevoj strani zaslona nalazi se traka za navigaciju (Slika 3.1) koja se na većim uređajima proširuje kako bi prikazala više informacija (Slika 3.2).

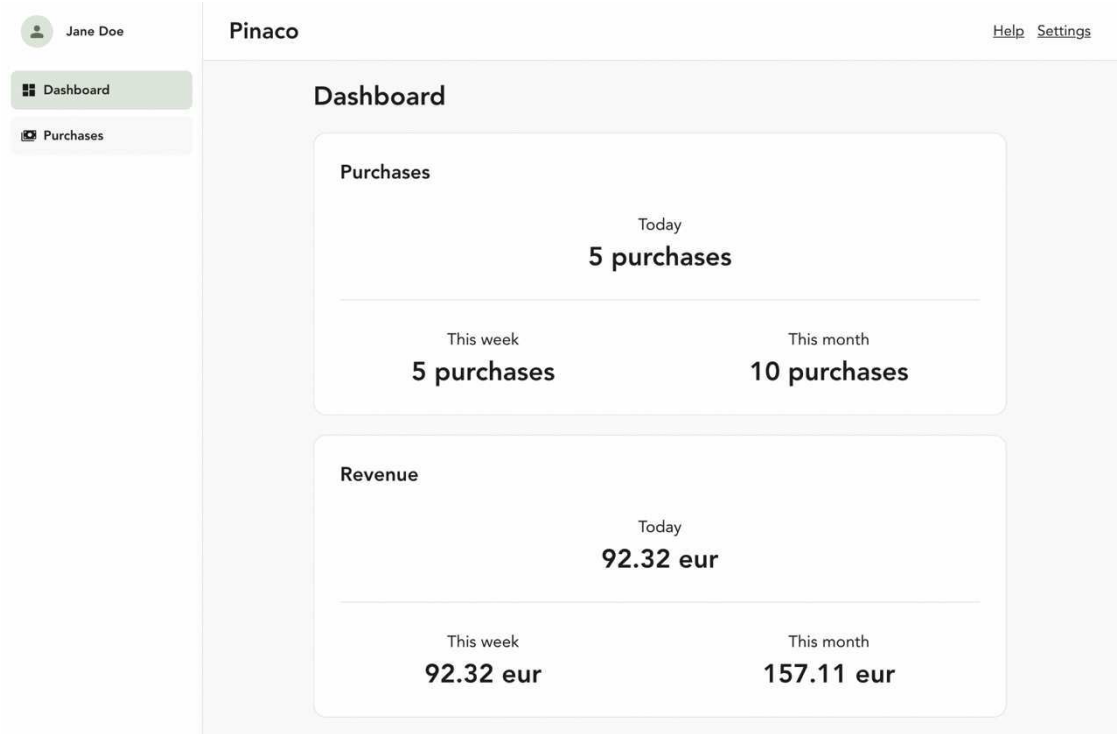


Slika 3.1 Navigacijska traka na malom zaslonu



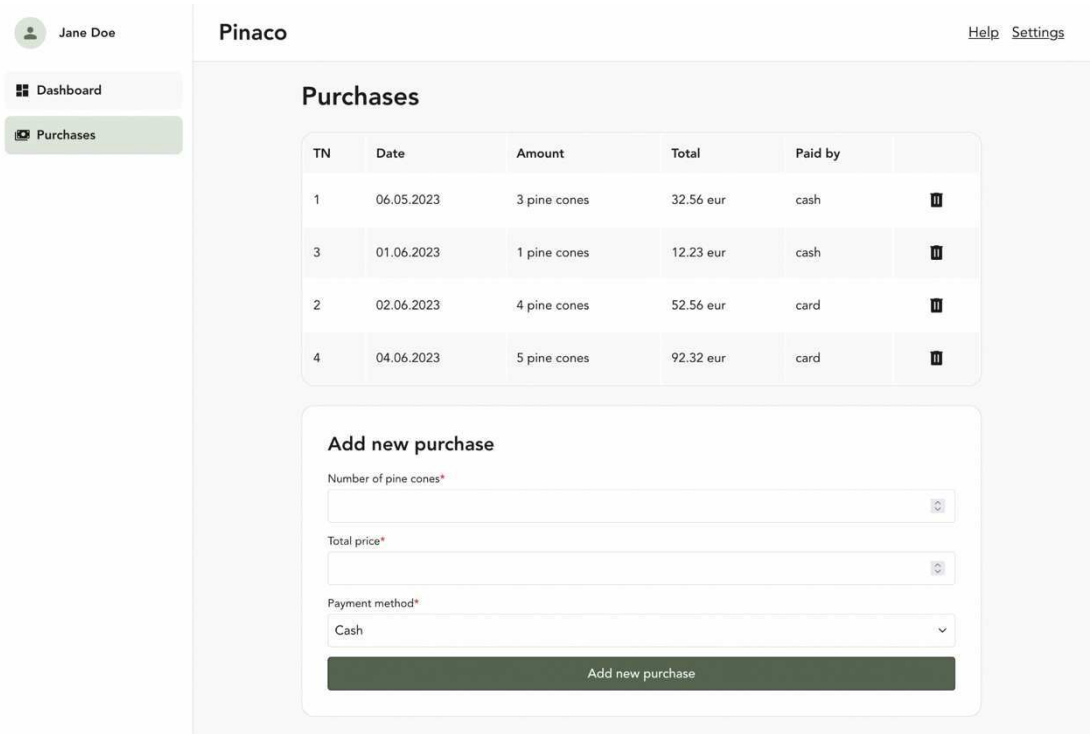
Slika 3.2 Navigacijska traka na velikom zaslonu

Prva stranica, *Dashboard*, pruža pregled nekoliko parametara koji se dohvaćaju s poslužiteljske strane (Slika 3.3), a računaju se na temelju podataka koji se unose i pregledavaju na stranici *Purchases*.



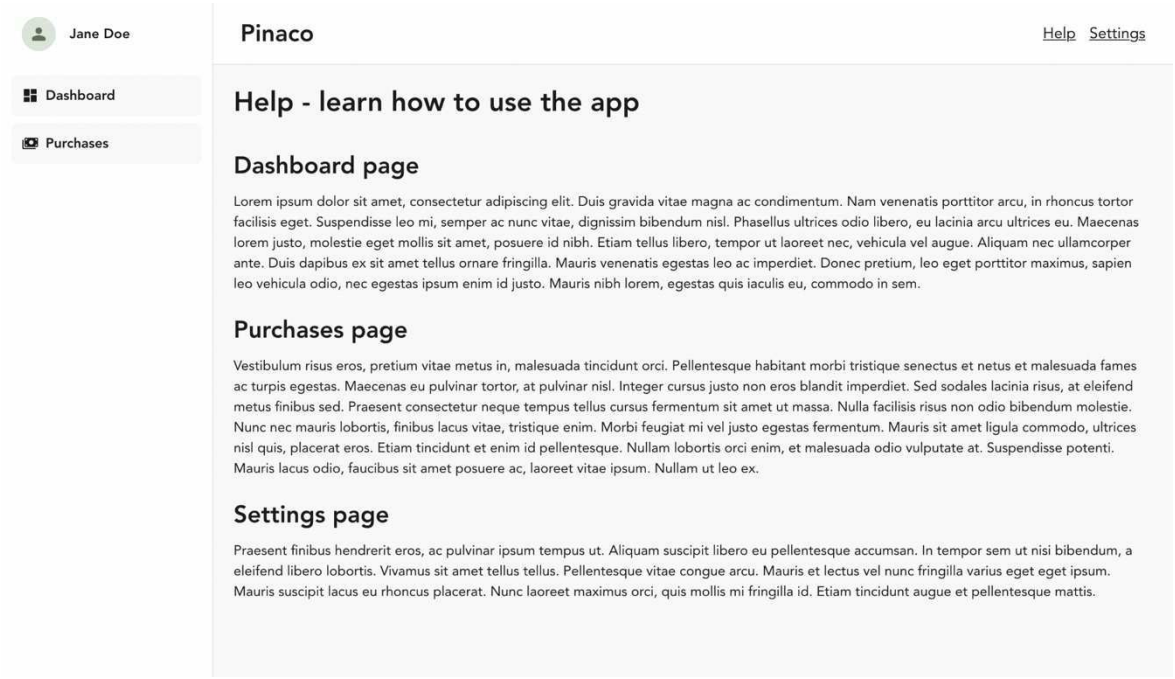
Slika 3.3 Stranica Dashboard dinamične web aplikacije

Druga stranica, *Purchases* (Slika 3.4) prikazuje popis podataka (popis svih zamišljenih transakcija koje su prikazane u ovoj aplikaciji) gdje svaka transakcija sadrži gumb za brisanje iste, a ispod popisa nalazi se i forma za upisivanje novih transakcija u sustav.



Slika 3.4 Stranica Purchases dinamične web aplikacije

Preostaje nam još i stranica *Help* (poveznica u zaglavlju) koja prikazuje potpuno statičan sadržaj (Slika 3.5), a ne donosi nikakve novosti od prethodne aplikacije pa je tu samo radi potpunosti. Gumb *Settings*, također u zaglavlju, vodi na nepostojeću stranicu, opet radi potpunosti aplikacije jer funkcionira na isti način kao u prethodnoj aplikaciji.



Slika 3.5 Stranica Help dinamične web aplikacije

Ova je aplikacija također potpuno responzivna i radi na svim veličinama uređaja. Ikone korištene u aplikaciji preuzete su s Google Icons.

3.1. Razvoj aplikacije

I ove aplikacije razvijene su koristeći knjižnicu styled-components, no za razliku od statične aplikacije, Reactova verzija ove aplikacije razvijena je koristeći dodatne knjižnice na mjestima na kojima Next.js pruža svoje rješenje. Tako su dodane i knjižnice wouter za navigaciju, swr za dohvat podataka i react-helmet za upravljanje naslovom koji se prikazuje u kartici web preglednika. Uz to, u obje aplikacije korištena je knjižnica dayjs radi lakše manipulacije i prikaza datuma.

Inicijalizacija projekata odrađena je na isti način kao i u statičnoj aplikaciji pa tu nema većih razlika.

Razvoj Reactove verzije aplikacije znatno su olakšale dodatne knjižnice, pa se iskustvo u razvoju praktički izjednačilo s razvojem u Next.js-u. Nažalost, budući da se i dalje radi o

aplikaciji koja se renderira na klijentu, moramo ručno prikazivati greške, a i stanje učitavanja i dalje postoji pa i za njega mora postojati dizajn sučelja, što zahtijeva dodatan kod.

Kada obavljamo akcije koje mijenjaju sadržaj na stranici, situacija je nešto drugačija. Budući da Next.js renderira stranicu na poslužitelju, da bismo primili ikakve izmjene podataka stranici moramo opet zatražiti Nextov poslužitelj da renderira stranicu pa nam vrati podatke, pa sve što moramo (a i možemo) jest nakon uspješnog obavljanja akcije pozvati `router.refresh()` iz `next/router` paketa

Koristeći neki paket za dohvat podataka na klijentu, u ovom slučaju *swr*, možemo učiniti isto - obrisati iz priručne memorije (koju implementira i njom upravlja sama knjižnica - swr) podatke za sve dohvate podataka na koje obavljena akcija utječe, ili jednostavnije, isprazniti cijelu priručnu memoriju. No, vrlo jednostavno možemo i modificirati priručnu memoriju tako da odgovara izmjenama, bez da tražimo poslužitelj za nove podatke. To dosta komplicira stvari, ali je mnogo efikasnije, pa možemo i kombinirati ove dvije taktike - to smo i učinili u ovoj aplikaciji - jednostavnije izmjene se obavljaju lokalno modificirajući priručnu memoriju, a za kompliciranije izmjene brišemo taj zapis u priručnoj memoriji. Tako na primjer, prilikom dodavanja zapisa o transakciji, poslužitelj vrati novi zapis, a mi ga na klijentu umetnemo u priručnu memoriju. Za stranicu *Dashboard* bismo trebali raditi kompliciranije izračune (za izračun bismo morali računati dan svake transakcije te grupirati transakcije prema tome) pa ćemo jednostavno te zapise izbrisati iz priručne memorije, a oni će se sami dohvatiti sljedeći puta kad posjetimo tu stranicu.

3.2. Krajnja aplikacija

U ovom poglavlju analizirat ćemo ovu aplikaciju na sličan način kao i statičnu aplikaciju, no na kraju ćemo usporediti i obavljanje akcija (mijenjanje podataka - dodavanje i brisanje). Ovaj puta ćemo u Reactovoj aplikaciji ručno razdvojiti kod - po rutama, slično načinu na koji to čini i Next.js. Prvo ćemo usporediti veličine JavaScriptovih paketa koje obje aplikacije dostavljaju na klijentsko računalo (prvo po rutama - /, /about, /help) (Tablica 3.1).

Tablica 3.1 Veličina završnih JavaScriptovih paketa dinamične aplikacije

	React	Next.js
/ - gzip	76 kB	96 kB
/purchases - gzip	80 kB	100 kB
/help - gzip	73 kB	96 kB
ukupno - gzip	83 kB	103 kB
ukupno - stvarna veličina	241 kB	321 kB

I ovdje React.js ima prednost, no uočimo značajnu razliku između stranica na adresama / i `/purchases` (*Dashboard* i *Purchases*). To je zbog toga što na stranici *Purchases* imamo mnogo više funkcionalnosti, pa tako i samog koda, no na toj stranici koristimo i vanjsku knjižnicu dayjs koja sama ima oko 6.5 kB [6].

Pogledajmo sad razliku kod obavljanja akcija. Uzmimo za primjer brisanje jedne od transakcija. Počevši s 10 transakcija, izbrisat ćemo jednu od transakcija, a zatim pratiti količinu podataka koje klijent mora slati ili primati od poslužitelja te broj zahtjeva s klijenta prema poslužitelju (Tablica 3.2).

Tablica 3.2 Veličina i broj zahtjeva na obavljanje akcije

	React.js	Next.js
Količina prenesenih podataka	216 B	771 B
Broj zahtjeva s klijenta prema poslužitelju	1 zahtjev	2 zahtjeva

Vidimo da Reactova aplikacija napravi jedan zahtjev prema poslužitelju: šalje ID transakcije koju treba obrisati. Nakon što poslužitelj odgovori pozitivno, aplikacija u svojoj priručnoj memoriji iz popisa transakcija miče tek obrisanu transakciju. S druge strane, Next aplikacija nema tu kontrolu, jer priručna memorija ne postoji već su podaci statično primljeni s poslužitelja. Zato aplikacija opet mora zahtijevati podatke (cijele stranice). U ovom slučaju nije mnogo podataka, ali kada bi popis sadržavao nešto više podataka ili kada bi se na stranici nalazili još neki podaci koji nisu vezani uz transakcije, količina podataka koji se bespotrebno dohvaćaju postala bi značajna. Podrazumijeva se, taj dodatni zahtjev stavlja pritisak i na poslužitelj.

Rezultate mjerenja u Lighthouseu ovdje ćemo razmotriti za najdinamičniju stranicu, *Purchases*. Kao i prethodno, razmotrit ćemo tri mjere: *First Contentful Paint*, *Largest Contentful Paint* i *Total Blocking Time* (Tablica 3.3).

Tablica 3.3 Rezultati dinamične aplikacije u Lighthouseu

	React.js	Next.js
FCP	1.3 s	0.7 s
LCP	1.9 s	0.8 s
TBT	140 ms	130 ms

I ovdje vidimo da je Next.js nešto brži, a i dalje aplikacije nisu dovoljno zahtjevne da bi blokirale glavnu dretvu dovoljno dugo da bismo to primijetili.

3.3. Zaključak

Za razvoj ovakve, dinamičnije web aplikacije, Next.js i dalje donosi određene prednosti, uglavnom u pogledu boljeg razvojnog iskustva. No, što je aplikacija dinamičnija, tj. što je više akcija u kojima se mijenjaju prikazivani podaci, to veću prednost ima čisti React. Nextov pristup, pogotovo na većim projektima, postaje vrlo neefikasan, a prednosti koje donosi više nisu toliko izražene. Većinu značajki (osim renderiranja na poslužitelju) moguće je lagano zamijeniti drugim (ponekad i boljim, ili barem boljima za određenu primjenu) knjižnicama.

Također, za ovakvu vrstu aplikacije nije nam toliko bitno da bude renderirana na poslužitelju, jer je ionako skrivena iz autentifikacije pa je internetski pretraživači ne mogu vidjeti. Brzina prvog učitavanja isto nema toliku važnost kao na nekoj stranici *landing page* - korisnici ovdje dolaze s namjerom, i neće tako lako odustati ako stranici treba koja sekunda više da se učita.

Ponekad ni razdvajanje koda nema smisla - ako je aplikacija namijenjena za korištenje na nekom kiosku ili na bilo kojem mjestu na kojem će se koristiti dugotrajno, bez zatvaranja, više je smisleno učitati cijelu aplikaciju odmah, nego ju učitavati dio po dio.

Zaključak

JavaScript je u web uveo velike napretke te promijenio način na koji ga koristimo. React.js i Next.js samo su dva od velikog broja alata koji nam pomažu u iskorištavanju svih mogućnosti koje JavaScript pruža da bismo stvorili interaktivne web aplikacije.

React.js, kao jedna od najučestalijih knjižnica na današnjem webu, omogućava nam lakšu, bržu i jeftiniju izgradnju velikih, interaktivnih web aplikacija. Next, s druge strane, pokušava proširiti ponudu Reacta na način da uvodi renderiranje na poslužitelju. Renderiranje na poslužitelju donosi velike prednosti za određene primjene, glavne od kojih su brže prvo učitavanje i bolje rangiranje na internetskim tražilicama poput Googlea. S druge strane, za neke primjene te dvije stvari nisu od velike važnosti, pa nam uvođenje renderiranja na poslužitelju bespotrebno komplicira razvoj i održavanje aplikacija. Potreba za poslužiteljem kod Next.js-a i dodatna ograničenja kod razvoja aplikacije nisu uvijek vrijedna malo bržeg prvog učitavanja, pogotovo kad rezultati internetskih tražilica ne nose veliku vrijednost.

Usporedili smo knjižnicu React.js s radnim okvirom Next.js u dvije različite aplikacije. Prvu smo razvili da bude gotovo statična, da imitira tipičnu web stranicu nekog proizvoda ili tvrtke. U Reactovoj verziji te aplikacije nismo koristili dodatne biblioteke, kako bismo vidjeli koliko pomoći nam Next pruža. U takvoj aplikaciji nam Nextove dodatne značajke uvelike olakšavaju razvoj, a i krajnji rezultat je bolji. Renderiranje na poslužitelju omogućava internetskim tražilicama da ispravno rangiraju našu web stranicu, a i brže učitavanje znači da će novi korisnici rjeđe odustati čekajući da se stranica učita.

Druga aplikacija je nešto dinamičnija, omogućava korisniku da čita i mijenja dinamične podatke. U Reactovoj verziji ove aplikacije rupe koje Next popunjava, popunili smo dodavanjem dodatnih knjižnica. Zbog toga je razvoj bio puno lakši, praktički na razini razvoja koristeći Next.js, a krajnja aplikacija znatno je efikasnija. Renderiranje na poslužitelju nam nije od neke koristi, jer je ovakva aplikacija tipično sakrivena iza autentifikacije, a i korisnici je posjećuju ciljano, s namjerom da odrade nešto, pa neće tako lako odustati ako moraju malo duže pričekati da se cijela aplikacija učita.

Međutim, nismo uzeli u obzir potporu koju Next.js pruža kao radni okvir. Next.js nameće nam određenu strukturu, koja, iako ograničava, ipak donosi i nekakav standard. Zbog toga,

Next.js bi mogao biti dobar odabir i manje stručnim programerima kojima treba radni okvir koji će ih voditi kroz razvoj aplikacije, ali i većim timovima kojima nedostaje struktura ili se jednostavno ne mogu dogovoriti oko praksi kojih će se držati.

Literatura

[1] Pixabay (2023. svibanj). Poveznica: https://pixabay.com/; pristupljeno 30. svibnja 2023.

[2] Lighthouse Overview, Chrome Developers, (svibanj 2023.). Poveznica: https://developer.chrome.com/docs/lighthouse/overview/; pristupljeno 30. svibnja 2023.

[3] First Contentful Paint, Chrome Developers, (svibanj 2023.). Poveznica: https://developer.chrome.com/docs/lighthouse/performance/first-contentful-paint/; pristupljeno 31. svibnja 2023.

[4] Largest Contentful Paint, Chrome Developers, (svibanj 2023.). Poveznica: https://developer.chrome.com/docs/lighthouse/performance/lighthouse-largest-contentful-paint/; pristupljeno 31. svibnja 2023.

[5] Total Blocking Time, Chrome Developers, (svibanj 2023.). Poveznica: https://developer.chrome.com/docs/lighthouse/performance/lighthouse-total-blocking-time/; pristupljeno 31. svibnja 2023.

[6] Bundlephobia, (lipanj 2023.). Poveznica: https://bundlephobia.com/package/dayjs@1.11.8; pristupljeno 4. lipnja 2023.

Sažetak

Cilj ovog završnog rada jest usporediti knjižnicu React.js za izradu dinamičnih web aplikacija i radni okvir Next.js, dodatak na knjižnicu React.js koji omogućava renderiranje aplikacije na poslužitelju. Izrađene su po dvije istovjetne aplikacije u obje tehnologije, a zatim uspoređen proces razvoja, kao i aplikacije koje su rezultat razvoja. Za statične aplikacije koje ne pružaju mogućnost izmjene podataka Next.js pokazao se kao bolji izbor jer pruža renderiranje na poslužitelju, koje je bitno radi boljeg rangiranja na internetskim tražilicama, ali i bolje performanse prilikom prvog učitavanja. Za aplikacije koje omogućavaju izmjenu podataka React se pokazao kao bolji izbor, jer unatoč nešto sporijem prvom učitavanju i nedostatku renderiranja na poslužitelju, React pruža mogućnost implementacije priručne memorije na klijentskoj strani, pa je nakon svake izmjene na poslužitelju moguće samo izmijeniti priručnu memoriju, dok je u Nextu potrebno opet dohvatiti sve podatke na trenutnoj stranici (pa čak i one koji se nisu promijenili). Također, takve su aplikacije uglavnom sakrivene iza autentifikacije pa renderiranje na poslužitelju i brzina prvog učitavanja nisu presudni.

Summary

The aim of this bachelor thesis is to compare the dynamic web front-end library React.js with its companion for rendering these web applications on the server side, Next.js. Two identical pairs of applications were built, then analysed along with the process of their development. It is concluded that Next.js is a better option for static websites that do not allow mutating data as it allows for server-side rendering, making these applications better with search engines, as well as having better first load speed. React, however, is a better choice for web applications that allow for mutating data, as despite its lack of server-side rendering and slower first load, it allows for implementing a frontend cache which can be easily updated after mutating data, whereas applications built with Next.js need to refetch all the displayed data, even if some of it might not have changed. These applications are usually protected by authentication as well, so server-side rendering and first load speed are not of such importance.