Izrada web aplikacije za organizaciju knjiga pomoću React biblioteke za Javascript

Vidiček, Marko

Sveučilište u Rijeci – Fakultet informatike i digitalnih tehnologija

Diplomski sveučilišni studij Informatika - Informacijski i komunikacijski sustavi (IKS)

Marko Vidiček

Izrada web aplikacije za organizaciju knjiga pomoću React biblioteke za Javascript

Diplomski rad

Mentor: Doc. dr. sc. Lucia Načinović Prskalo

Rijeka, Rujan 2022.

FAKULTET INFORMATIKE I DIGITALNIH TEHNOLOGIJA
SVEUČILIŠTE U RIJECI

uniri

Rijeka, 17. Kolovoza 2022.

Zadatak za diplomski rad

Pristupnik: Marko Vidiček

Naziv diplomskog rada: Izrada web aplikacije za organizaciju knjiga pomoću React biblioteke za Javascript.

Naziv diplomskog rada na eng. jeziku: Developing a bookshelf web application using React framework for Javascript.

Sadržaj zadatka: Zadatak ovog diplomskog rada je proučiti moderne tehnologije za izradu web stranica (React, Next.js, MantineUI) i opisati izradu web aplikacije za pretragu i organizaciju knjiga u web pregledniku korištenjem istih tehnologija.
Za programiranje web aplikacije se koristi Javascript programski jezik uz HTML i CSS te dodatne biblioteka.
Potrebno je opisati korištene tehnologije te njihove funkcionalnosti i uloge u izradi web aplikacije.

Mentor:
Doc. sr. sc. Lucia Načinović Prskalo

Voditeljica za diplomske radove:
Prof. dr. sc. Ana Meštrović

Zadatak preuzet: 26.4.2022.

Marko Vidiček
(Potpis Pristupnika)

Fakultet informatike i digitalnih tehnologija
Radmile Matejčić 2, 51000 Rijeka, Hrvatska
http://inf.uniri.hr ured@inf.uniri.hr
Tel: +385 (0)51584700 | Fax: +385 (0)51 584 749

uniri | YUFE

Sažetak

Ovaj završni rad opisuje proces izrade web aplikacije pod nazivom “Biblio”.

Web aplikacija omogućuje korisniku pretraživanje knjiga uz pomoć Google Books API-a (U budućnosti će podržavati i Open Library API), te prikazivanje detalja o istim knjigama i organizaciju u kolekcije koje sadrže one knjige koje korisnik trenutno čita, koje je pročitao i koje je označio da ga zanimaju (Bookmarks).

U budućnosti će aplikacija podržavat i dodatne funkcionalnosti poput filtera za parametre upita koji se šalju na API endpoint, kreiranje izazova za čitanje, prikaz statistika korisnika, promjena izgleda i postavka aplikacije te korisničkih podataka.

Tehnologije koje se koriste u izradi web aplikacije su

- **React** (JavaScript biblioteka za izradu korisničkih sučelja)
- **Next.js** (React Framework za proizvodnju)
- **MantineUI** (Biblioteka React UI komponenti)
- **Google Firebase** (Platforma za razvoj aplikacija, omogućuje autentifikaciju i noSQL bazu podataka za backend)
- **SWR** (React hook funkcija za dohvaćanje podataka)
- **GitHub i Git** (Kontrola verzioniranja i remote repozitorij izvornog koda)
- **Netlify** (usluge poslužitelja)

Dodatne biblioteke

- *radix-ui/react-icons (Ikone)*
- *Lottie-react (animacije)*
- *Date-fns (formatiranje datuma)*
- *React-flags-select (odabir jezika)*

Ostale potrebne tehnologije uključuju GitHub za remote repozitorij koda te Node.js i NPM za pokretanje Javascript koda i dohvaćanje Javascript paketa.
Tehnologije će biti detaljno opisane u daljnjim poglavljima ovog diplomskog rada.

Prilikom izrade aplikacije većinom su korištene dokumentacije navedenih alata kao izvor informacija uz video sadržaje i članke na internetu.

Ključne riječi: Biblio, Web Application, Javascript, React, Next.js, Mantine, Firebase, SWR, WebStorm, HTML, CSS, GitHub, NPM, Node.js, Google Books API, Bookshelf

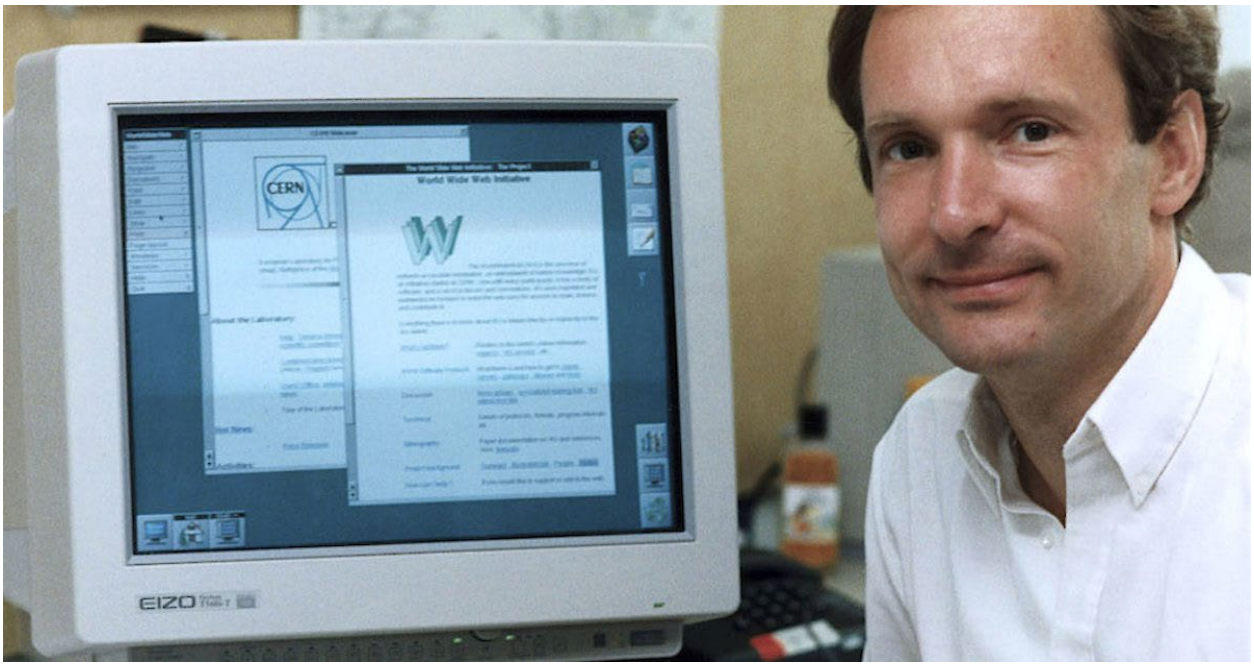
1. Uvod

1.1. Kratka povijest web razvoja

Znanje o povijesti web razvoja **[1]** će nam pomoći da bolje shvatimo web i njegovu revoluciju od starih dana do danas.

1993. Tim Berners-Lee napisao je prvu verziju HTML-a. Bio je fizičar i informatičar koji je implementirao prvu uspješnu komunikaciju između Hypertext Transfer Protocol (HTTP) klijenta. On je zapravo izumitelj interneta kakvog danas poznajemo i volimo.

HTML je imao mnogo verzija tijekom sljedećih nekoliko godina. HTML v2 objavljen je 1995., nakon čega su uslijedili v3 i v4 1997. Na kraju, to nas dovodi do HTML5, koji je objavljen 2011.



Slika 1. Tim Berners-Lee

Tijekom prve godine, HTML je izgledao prilično nezgrapno. No, sve se to počelo mijenjati krajem 1994. kada je Håkon Wium Lie objavio prvi nacrt prijedloga Cascading HTML Style Sheets (CSS).

Prvo izdanje CSS-a izazvalo je malu dramu. Mnogi su ga smatrali prejednostavnim za zadatak za koji je dizajniran. Tvrdilo se da je za stiliziranje dokumenata potrebna snaga potpunog programskog jezika.

Nakon godinu dana, CSS je postao preporuka World Wide Web Consortium-a (W3C). Godine 1997. CSS v2 objavljen je i šire se koristio u preglednicima, uključujući Netscape Navigator.

Godine 1995. programer Netscape-a po imenu Brandan Eich razvio je novi skriptni jezik pod nazivom Mocha, koji je s vremenom postao poznat kao JavaScript.

To je kratka povijest čvrstih temelja (HTML, CSS, JavaScript) koje danas koristimo za svaku web aplikaciju.

Naravno, ovi jezici ne mogu učiniti mnogo bez skriptnog jezika na strani poslužitelja **[2]** koji ide uz njih.

U ranim danima HTML-a, odgovori poslužitelja slani su iz Shell, Python ili C skripte.

Godine 1994. Rasmus Lerdorf napisao je skup binarnih datoteka Common Gateway Interface (CGI) napisanih u C programskom jeziku.

To je omogućilo Rasmusu da stvori jednostavnu skriptu na strani poslužitelja za svoju "Osobnu početnu stranicu", kada je rođen programski jezik PHP.

PHP je nastavio rasti u popularnosti, a korišten je za stvaranje popularnog CMS-a, Wordpressa. PHP je također korišten za stvaranje mnogih popularnih web stranica, uključujući Facebook.

Došlo je i do malog porasta popularnosti ASP-a, koji je Microsoft na kraju usvojio i koristio za izradu ASP.net i C# za pisanje aplikacija na strani poslužitelja.

Godine 2005. Ruby je postao još jedna zvijezda u usponu, točnije Ruby On Rails (RoR), koju je stvorio David Heinemeier Hansson.

Ruby on Rails bio je okvir za izradu web aplikacija, jednostavan za korištenje i izgrađen na temelju programskog jezika Ruby. Ovo je pokrenulo novu generaciju nezavisnih web programera i kreatora web aplikacija.

Otprilike u isto vrijeme, 2005. godine, novi okvir Django bio je sve popularniji. Ovo je unaprijedilo upotrebu Python-a kao jezika na strani poslužitelja. Python se danas naširoko koristi za mnoge web aplikacije.

Zatim, 2009., Ryan Dahl je stvorio način za pisanje JavaScripta koji bi ga izvršavao kao jezik poslužitelja. Ovaj programski jezik poznat je kao NodeJS i brzo je postao popularan i naširoko korišten.

1.2. Web aplikacije

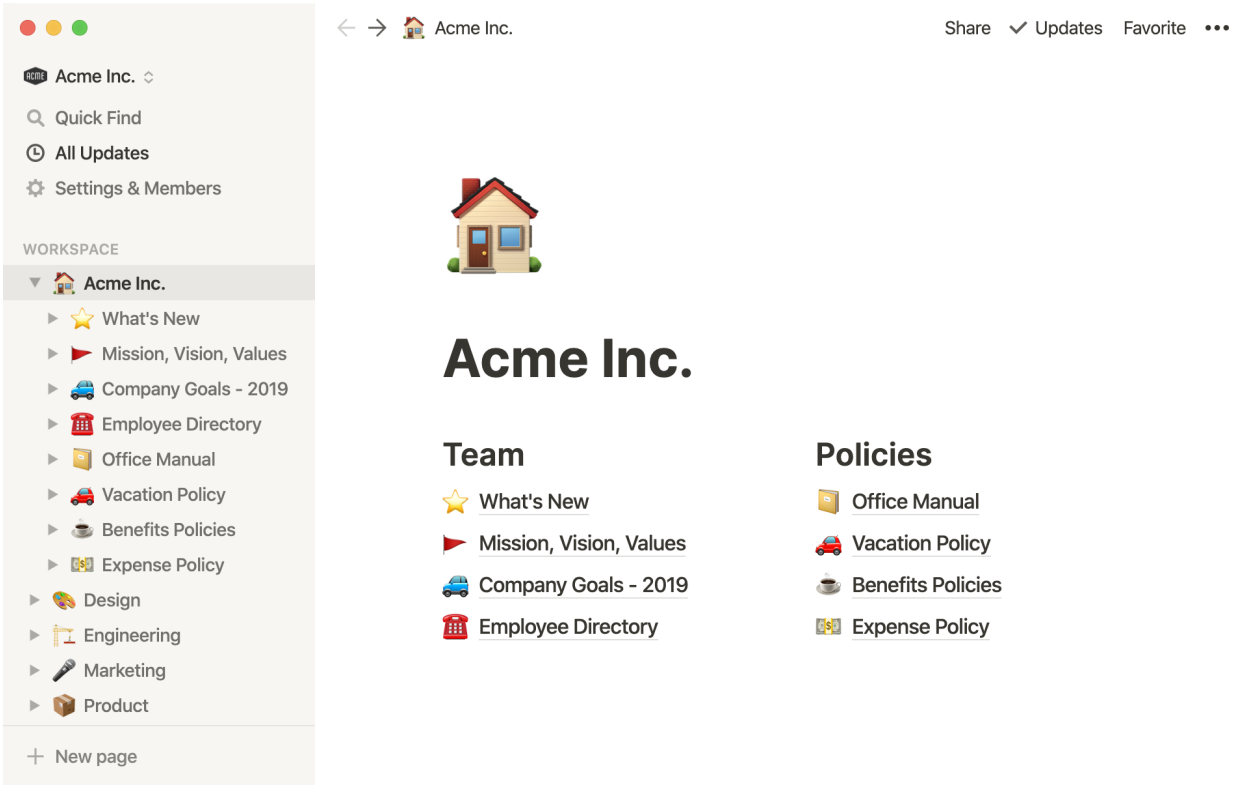
Web aplikacija **[3]** je interaktivni računalni program izgrađen uz pomoć web tehnologija (HTML, CSS, JS), koji pohranjuje (Baza podataka, Datoteke) i manipulira podacima (CRUD), a koristi ga tim ili jedan korisnik za obavljanje zadataka preko interneta.

CRUD je popularna kratica i u središtu je razvoja web aplikacija. Skraćenica je za stvaranje, čitanje, ažuriranje i brisanje. Web aplikacijama se pristupa putem web preglednika kao što je Google Chrome i često uključuje mehanizam prijave/registracije.

Ključna razlika između web stranica i aplikacija je u komunikaciji. Web aplikacije definirane su njihovim unosom – mi stvaramo, čitamo, ažuriramo i brišemo podatke unutar web aplikacije. Web stranice su definirane njihovim rezultatima - čitamo vijesti, marketinške informacije, FAQ na web stranicama.

Primjeri web aplikacija

- **Google Docs** (stvaranje, čitanje, ažuriranje i brisanje dokumenata)
- **Notion** (Web aplikacija za organizaciju bilješki i suradnju s podrškom za markup jezik)



Slika 2. Notion, Web Aplikacija

1.2.1. Prednosti

Nevjerojatno je **jednostavno pokrenuti** web aplikaciju. Postoji vrlo malo obruča kroz koje moramo preskočiti u usporedbi sa desktop i mobilnim aplikacijama, kao i puno više fleksibilnosti u smislu alata i okvira koje možemo koristiti.

Kako bismo svoju aplikaciju uživo prikazali korisnicima, samo im treba poslati URL.

Slično tome, razvoj web aplikacija je pametniji izbor ako želimo korisnicima **olakšati pronalaženje i korištenje** alata. Danas većina korisnika radi sve iz svojih web preglednika - čak i u profesionalnim kontekstima.

Ljepota web aplikacija je u tome što im korisnici mogu pristupiti iz bilo kojeg web preglednika.

Preuzimanje novih aplikacija može biti nezgodno. Isto vrijedi i za pretrpan početni zaslon na telefonu i kraće trajanje baterije. Uglavnom korisnici neće preuzeti vašu aplikaciju osim ako to nije nešto što će koristiti gotovo svaki dan.

Razvoj web aplikacija također je **jeftiniji i brži** od izrade izvornih aplikacija ili programa za stolna računala. Osim toga, razvoj web aplikacija obično zahtijeva znatno manje prilagođenog rada nego druge vrste softvera.

Dio toga dolazi iz sveprisutnosti različitih okvira, frontend biblioteka i drugih alata koji ubrzavaju razvoj.

1.2.2. Nedostatci

Općenito govoreći, iako ne uvijek, web aplikacije će od korisnika zahtijevati **stabilnu internetsku** vezu. U najmanju ruku, obično će morati biti na mreži kako bi dobili punu funkcionalnost.

Uz sve ostale stvari, web aplikacije također će imati određena **funkcionalna ograničenja**, posebno u pogledu hardvera i drugih izvornih značajki na određenim uređajima.

Klasičan primjer za to bila bi uporaba kamere ili mikrofona na određenim uređajima, ali u današnje vrijeme različiti operacijski sustavi to prilično olakšavaju korisnicima.

1.3. Procesi razvoja web aplikacije

Faza razvoja za izradu web aplikacija podijeljena je u tri dijela kako bi se eliminirali pogrešni postupci i ubrzao proces web razvoja.

1.3.1. Dizajn

U fazi projektiranja UI dizajner, frontend web programer, poslovni analitičar i klijent zajednički odlučuju o shemi boja i početnoj ideji izgleda web aplikacije uzimajući u obzir funkcionalnosti koje mora izgraditi frontend web programer.

Web dizajner dizajnira okvire i modele koje zatim koriste frontend web programeri i UX dizajneri kao nacrt za stvaranje korisničkog sučelja web aplikacija pomoću HTML, CSS i JavaScript koda.

1.3.2. Frontend

Frontend **[4]** uključuje UX dizajn, analitičke animacije, upravljanje sesijom pomoću kolačića, implementaciju frontend sigurnosti i autentifikacije korisnika, kreiranje frontend funkcionalnosti, API integraciju, API pozive, usmjeravanje, rukovanje odgovorom web aplikacija.

Frontend web programeri imaju mnogo opcija za stvaranje interaktivnosti s JavaScriptom pomoću okvira. Web aplikacije uglavnom se razvijaju korištenjem frontend razvojnih okvira za web koji se temelje na MVC-u (Model-Prikaz-Kontroler arhitektura) koji kod frontend čini gotovo neovisnim o backendu. Frontend vidljiv je korisniku u opciji prikaza izvora web preglednika (inspect).

1.3.3. Backend

Backend **[4]** web stranice je mjesto koje sadržava sve podatke i relevantne informacije koje se trebaju prikazati posjetiteljima uz pomoć preglednika. Sučelje web stranice samo je način na koji se informacije prezentiraju korisnicima, a ono dohvaća sve iz pozadine za prikaz u korisničkim preglednicima.

Backend web razvoj uključuje upravljanje uslugama u pozadini, rukovanje bazom podataka, upravljanje sesijama, stvaranje sigurnosti za web aplikacije, razvoj API-ja.

2. Tehnologije

Kako bi bolje razumjeli praktični dio izrade web aplikacije, prvo je potrebno proučiti sve tehnologije koje se koriste u izradi web aplikacije “Biblio” i što nam svaka omogućuje u tom procesu.

Tehnologije koje se koriste i koje su opisane u ovome poglavlju su redom:

- Javascript - https://en.wikipedia.org/wiki/JavaScript
- Node.js - https://nodejs.org/en/about/
- NPM - https://www.npmjs.com
- Google Firebase - https://firebase.google.com
- React - https://reactjs.org
- Next.js - https://nextjs.org
- Mantine - https://mantine.dev
- SWR - https://swr.vercel.app
- GitHub i Git - https://github.com/about/
- Netlify - https://www.netlify.com/about/
- Dodatne biblioteke

2.1. Javascript

JavaScript **[5]** je programski jezik koji web programeri obično koriste za stvaranje dinamičnijih interakcija pri razvoju web stranica, aplikacija, poslužitelja ili čak igara.

Programeri općenito koriste JavaScript uz HTML i CSS. Skriptni jezik dobro radi s CSS-om u oblikovanju HTML elemenata. Međutim, JavaScript i dalje održava interakciju s korisnikom, nešto što CSS ne može učiniti sam.

Sam Javascript je relativno kompaktan, ali vrlo fleksibilan. Programeri su napisali razne alate povrh temeljnog JavaScript jezika, otključavajući golemu količinu funkcionalnosti uz minimalan napor. To uključuje:

- programska sučelja (API) ugrađena u web preglednike, koja pružaju funkcionalnosti kao što je dinamičko stvaranje HTML-a i postavljanje CSS stilova,

- API-ji koji programerima omogućuju ugradnju funkcionalnosti u web stranice drugih pružatelja sadržaja, kao što su Twitter ili Facebook,

- okviri i biblioteke koje se mogu primijeniti na HTML kako bi ubrzali rad na izgradnji stranica i aplikacija.

```
<script type="text/javascript" src="https://www.google.com/jsapi"></
  script>
<script type="text/javascript">

var bid = 43;
var ask = 21;


  google.load("visualization", "1", {packages:["corechart"]});
  google.setOnLoadCallback(drawChart);
  function drawChart() {
    var data = google.visualization.arrayToDataTable([
      ['Price', 'Quantity'],
      ['Value #1', bid],
      ['Value #2', ask],
    ]);
```

Slika 3. Javascript kod

2.2. Node.js

Node.js **[6]** je open-source okruženje za izvršavanje JavaScripta na više platformi koje se koriste za izvršavanje JavaScript koda izvan web preglednika. Koristi Google V8 Engine (https://v8.dev).

Node.js nam omogućuje korištenje JavaScripta svugdje i na bilo kojem pregledniku, uključujući MacOS, Linux i Windows. Kad kažemo posvuda, mislimo na frontend, middle-ware i back-end.

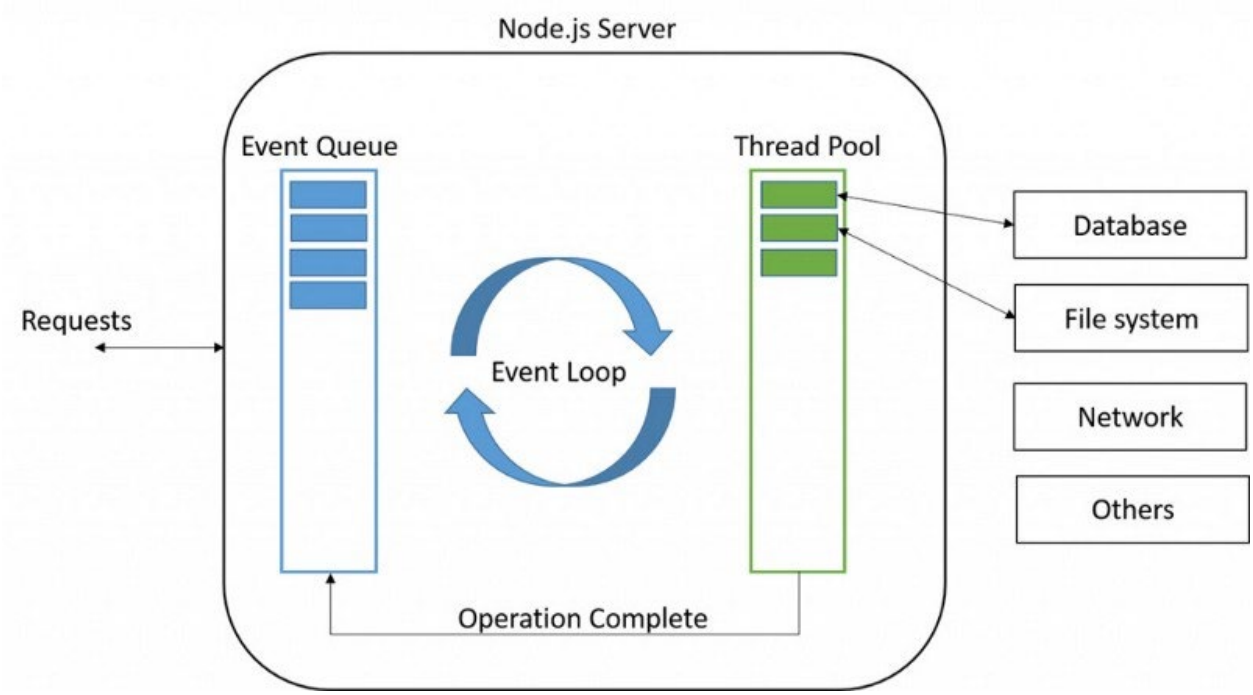
Kratak sažetak bi bio:

- Node.js je poslužiteljski framework i besplatan je,
- radi na Windowsima, Linuxu, Mac OS-u i drugima,
- Node.js koristi JavaScript na poslužitelju.

Node.js je single-threaded (jednonitna) platforma vođena događajima koja može pokretati neblokirajuće, asinkrono programiranje. Ove funkcionalnosti Node.js-a čine ga memorijski učinkovitim.

Petlja događaja ili Event Loop (Slika 4.) omogućuje izvođenje neblokirajućih I/O operacija unatoč činjenici da JavaScript radi na jednoj niti. To se radi dodjeljivanjem operacija operativnom sustavu kad god i gdje god je to moguće.

Većina operativnih sustava ima više niti i stoga može podnijeti više operacija koje se izvode u pozadini. Kada je jedna od ovih operacija dovršena, kernel obavještava Node.js i odgovarajući povratni poziv dodijeljen toj operaciji dodaje se u red događaja koji će se na kraju izvršiti.



Slika 4. Node.js, Event loop

(https://www.pabbly.com/tutorials/node-js-event-loops/)

2.3. NPM

Uz Node Package Manager (NPM) **[7]** programeri mogu otkriti i instalirati pakete koda u svoje mrežne aplikacije ili projekte na strani poslužitelja.

Node.js paket je direktorij s jednim ili više JavaScript modula ili biblioteka koji se koriste za dodavanje raznih značajki aplikacijama ili skriptama. Bez paketa, programer ili softverski inženjer mora napisati novi kod za svaku funkcionalnost koja je potrebna njihovom projektu.

Svaki npm projekt sadržava package.json, datoteku koja se nalazi u korijenskom direktoriju. Ta datoteka sadržava metapodatke npm projekata ili paketa, kao što su verzije paketa i suradnici.

Datoteka package.json pojednostavljuje prepoznavanje, upravljanje i instaliranje paketa. Zato je bitno uključiti package.json prije objavljivanja projekata u npm registru.

Primjer jednog zapisa paketa u package.json datoteci:

```
"dependencies": {
    "@mantine/core": "^5.1.4",
  },
```

`"Dependencies"` - Dependencies i devDependencies su polja u package.json datoteci koja navode sve pakete o kojima ovisi projekt.
Polje ovisnosti uključuje sve pakete trećih strana koji su potrebni za rad projekta.
S druge strane, devDependency sadržava pakete koji su potrebni samo tijekom razvoja.

`"@mantine/core"` - Naziv paketa koji je instaliran

`"^5.1.4"` - Verzija instaliranog paketa

2.4. Google Firebase

Google Firebase **[8]** je skup alata za "izgradnju, poboljšanje i razvoj aplikacija", a alati koje nudi pokrivaju veliki dio usluga koje bi programeri inače morali sami izgraditi, ali zapravo ne žele graditi.
To uključuje stvari kao što su **analitika**, **autentifikacija**, **baze podataka**, **konfiguracija**, **pohrana datoteka**, **slanje poruka** itd. Usluge se nalaze u oblaku i skaliraju se uz malo ili nimalo truda razvojnog programera.

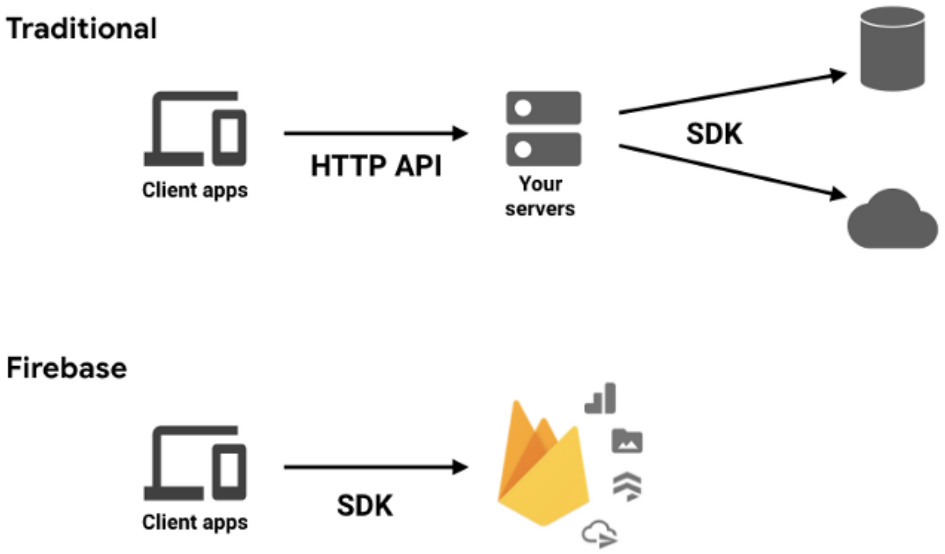
Usluge imaju pozadinske komponente koje u potpunosti održava Google. Klijentski SDK-ovi koje pruža Firebase izravno komuniciraju s tim pozadinskim uslugama, bez potrebe za uspostavom međuprograma između aplikacije i usluge.
Dakle, ako se koristiti Firebase baza podataka, obično se piše kod za postavljanje upita bazi podataka u klijentskoj aplikaciji (Slika 5.).

Ovo se razlikuje od tradicionalnog razvoja aplikacija, koji obično uključuje pisanje i frontend i backend softvera.

Sučelje samo poziva krajnje točke API-ja koje otkriva backend, a backend zapravo obavlja sav posao. Međutim, s Firebase proizvodima, tradicionalni backend je zaobiđen, prebacujući te funkcionalnosti na klijenta.
Administrativni pristup svakom od ovih proizvoda omogućuje Firebase konzola.

Usluge za izgradnju aplikacija koje nudi Firebase:

- Autentifikacija — prijava i identitet korisnika
- Cloud Firestore — realtime baza podataka, u oblaku, NoSQL
- Cloud Storage — masovno skalabilna pohrana datoteka
- Funkcije u oblaku
- Firebase Hosting — globalni web hosting
- ML Kit —SDK za uobičajene Machine Learning zadatke



Slika 5. Google Firebase

(https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0)

2.5. React

React **[9]** je JavaScript biblioteka koja se obično koristi za razvoj softvera koji stalno osvježava podatke na svom korisničkom sučelju. Razvio ga je Facebook.

Ova tehnologija eliminira potrebu ponovnog učitavanja cijelog zaslona i također izbjegava obradu svake pojedine linije koda.

React omogućuje stvaranje komponenti koje su zapravo napravljene s JavaScriptom i pruža korisne funkcije za izradu korisničkog sučelja, ali prepušta razvojnom programeru gdje će koristiti te funkcije u svojoj aplikaciji.

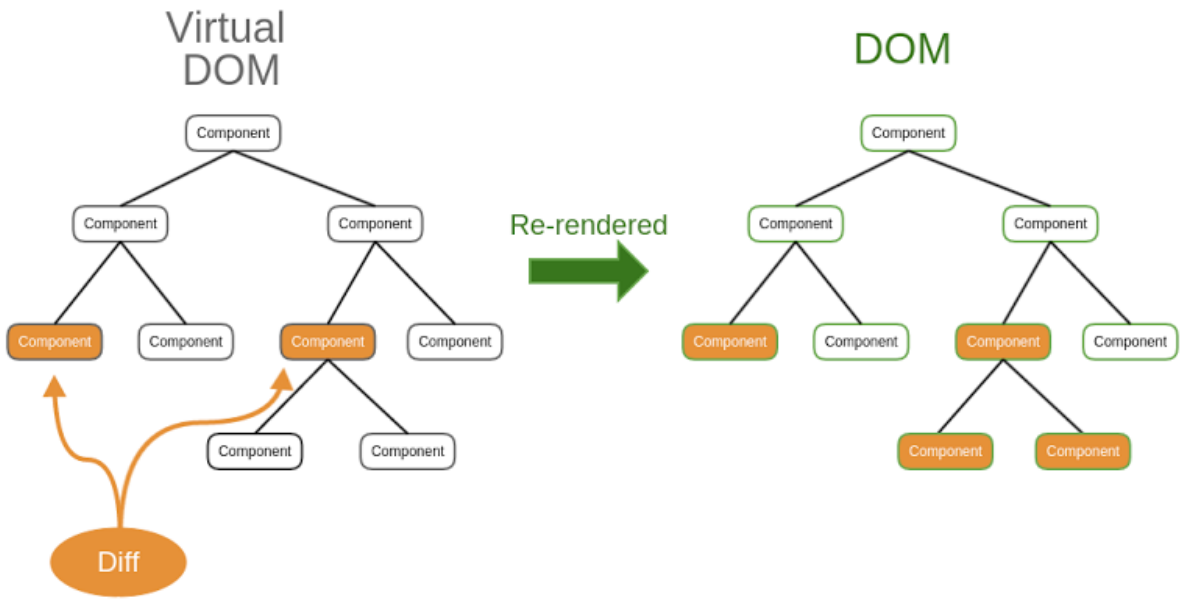
React koristi nekoliko ekstenzija pod nazivom JSX i Virtual DOM (Slika 6.) za stvaranje korisničkih sučelja. Oni omogućuju programerima da ih stvore za većinu platformi, a zahvaljujući činjenici da mogu odmah vidjeti rezultate svog koda, mogu imati bolji pregled i razumijevanje onoga što rade.

JSX je ekstenzija Reacta koja dopušta upotrebu HTML-a s JavaScriptom.

Virtualni DOM: DOM je skraćenica za Document Object Model, što je sučelje aplikacijskog programa (API). Omogućuje programima čitanje sadržaja bilo koje web stranice kako bi se mogao mijenjati prema željama i potrebama kodera. Svaka web stranica koja ne koristi React, koristi HTML za promjenu i modificiranje svog DOM-a.

Zahvaljujući implementaciji JSX-a React JS može imati Virtualni DOM, koji je kopija originalnog DOM-a koji koristi aplikacija ili web stranica.

Glavna razlika između DOM-a i Virtualnog DOM-a je u tome što prvi prikazuje promjene samo nakon ponovnog učitavanja stranice, dok ih drugi prikazuje u stvarnom vremenu bez potrebe ponovnog učitavanja.



Slika 6. React Virtual DOM

(https://medium.com/naukri-engineering/naukriengineering-virtual-dom-fa8019c626b)

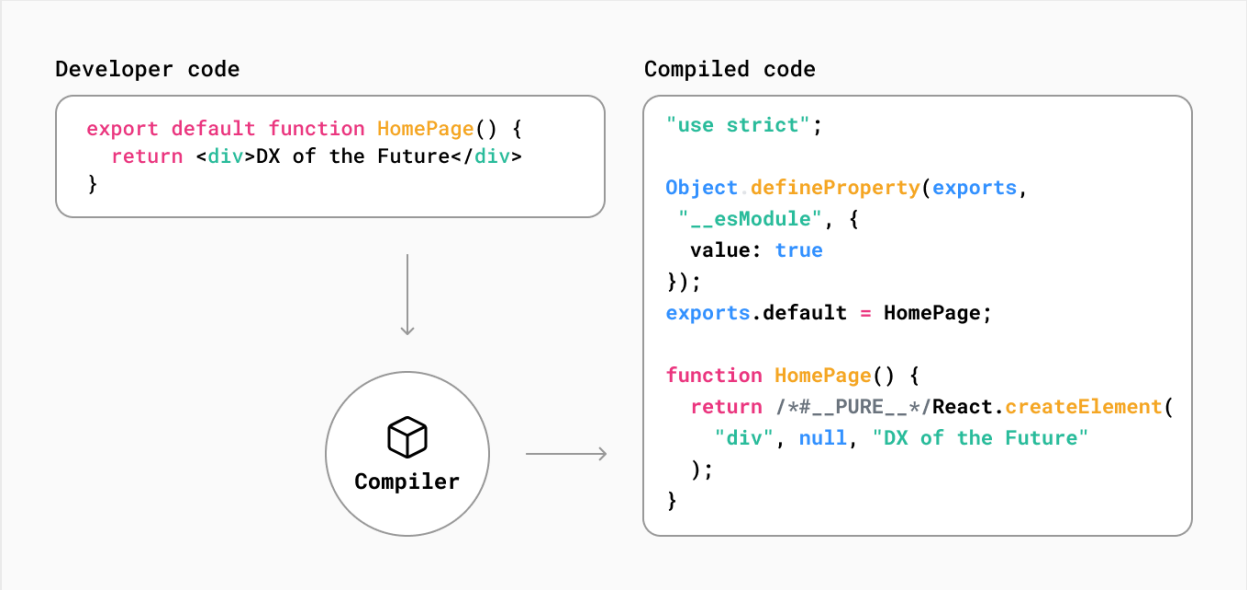
2.6. Next.js

Next.js **[10]** pruža značajke za fazu razvoja i produkcije aplikacije. Na primjer:
U fazi razvoja, Next.js optimizira iskustvo programera u izgradnji aplikacije. Dolazi sa značajkama koje imaju za cilj poboljšati razvojno iskustvo kao što su TypeScript i ESLint integracija, brzo osvježavanje i više.

U fazi proizvodnje, Next.js **optimizira aplikaciju** za krajnje korisnike i njihovo iskustvo korištenja aplikacije. Cilj mu je transformirati kod kako bi bio učinkovit i pristupačan.

Budući da svako okruženje ima drugačija razmatranja i ciljeve, puno je toga potrebno učiniti kako bi se aplikacija premjestila iz razvoja u proizvodnju. Na primjer, kod aplikacije treba kompajlirati (**compile**), spojiti (**bundle**), smanjiti **(minify**) i podijeliti na manje dijelove (**code-split**).

Programeri pišu kod na jezicima koji su prikladniji za programere, kao što su JSX, TypeScript i moderne verzije JavaScripta. Dok ti jezici poboljšavaju učinkovitost i samopouzdanje programera, potrebno ih je **prevesti** (Slika 7.) u JavaScript prije nego što ih preglednici mogu razumjeti.

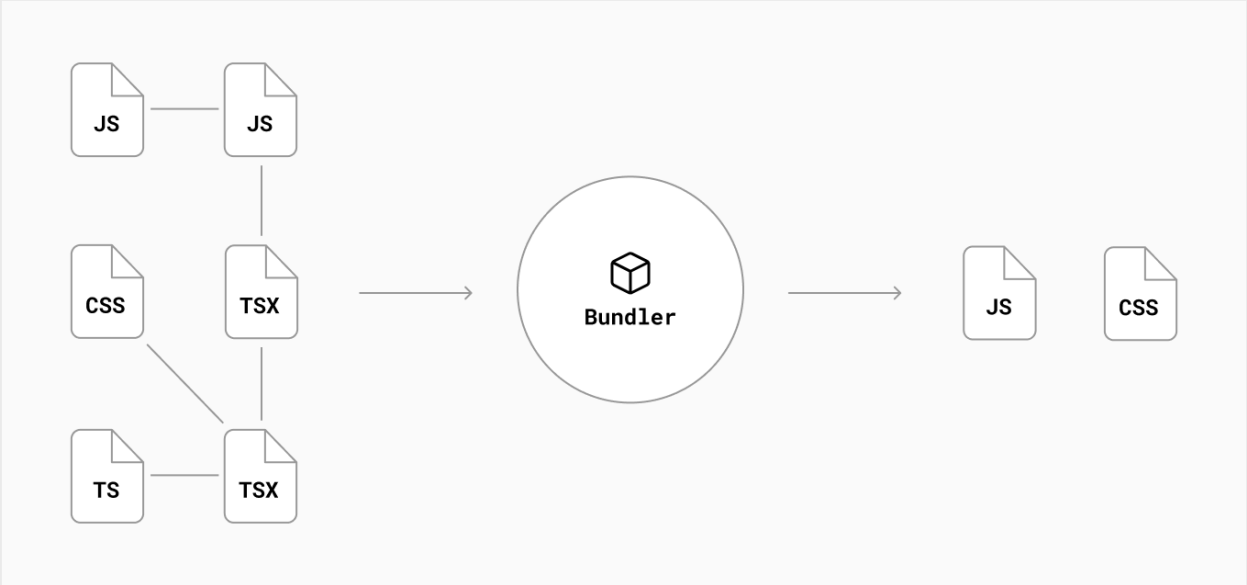


Slika 7. Next.js, Kompajliranje

(https://nextjs.org/learn/foundations/how-nextjs-works/compiling)

Programeri dijele svoje aplikacije na module, komponente i funkcije koje se mogu koristiti za izradu većih dijelova njihove aplikacije. Izvoz i uvoz ovih internih modula, kao i vanjskih paketa trećih strana, stvara složenu mrežu ovisnosti o datotekama.

Grupiranje (bundling) (Slika 8.) je proces rješavanja mreže ovisnosti i spajanja (ili 'pakiranja') datoteka (ili modula) u optimizirane pakete za preglednik, s ciljem smanjenja broja zahtjeva za datotekama kada korisnik posjeti web stranicu.

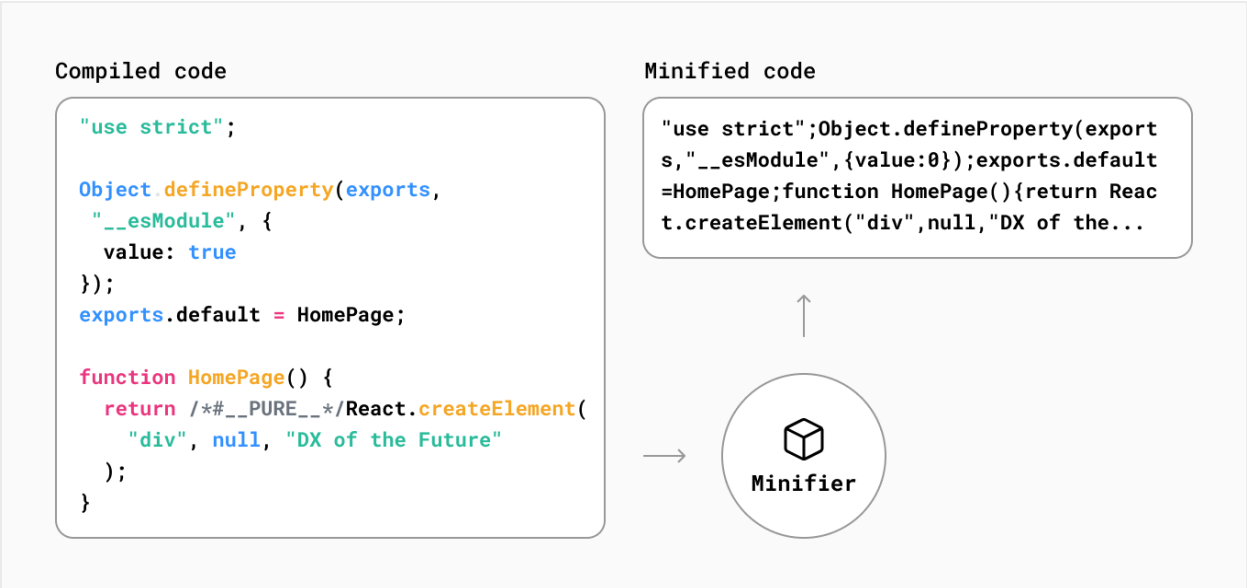


Slika 8. Next.js, Grupiranje

(https://nextjs.org/learn/foundations/how-nextjs-works/bundling)

Programeri pišu kod koji je optimiziran za čitljivost ljudi. Ovaj kod može sadržavati dodatne informacije koje nisu potrebne za izvođenje koda, kao što su komentari, razmaci, uvlake i više redaka.

Minificiranje (Slika 9.) je postupak uklanjanja nepotrebnog oblikovanja koda i komentara bez mijenjanja funkcionalnosti koda. Cilj je poboljšati performanse aplikacije smanjenjem veličine datoteka.

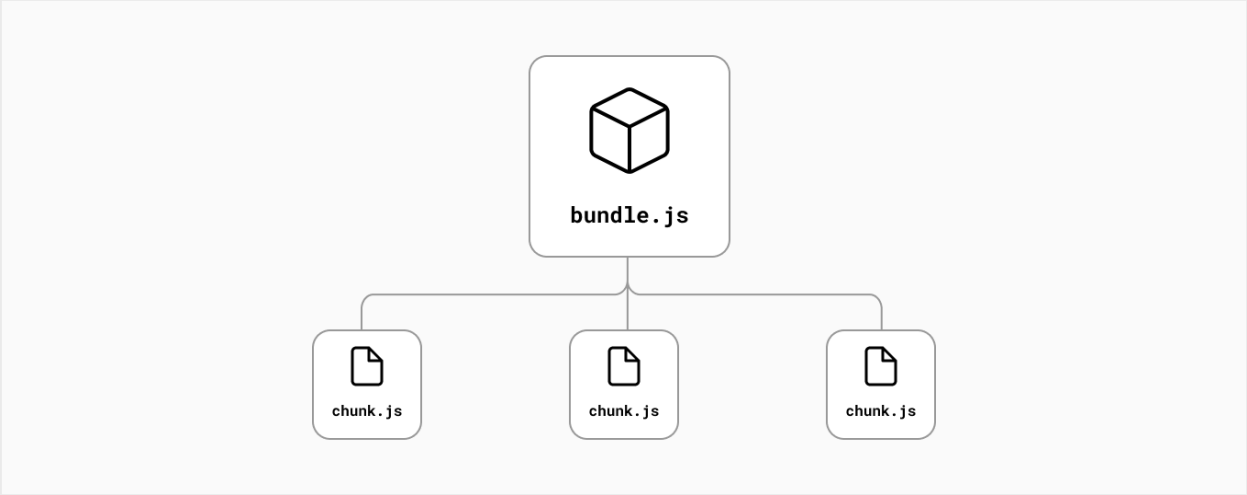


Slika 9. Next.js, Minificiranje

(https://nextjs.org/learn/foundations/how-nextjs-works/minifying)

Programeri obično dijele svoje aplikacije na više stranica kojima se može pristupiti s različitih URL-ova. Svaka od ovih stranica postaje jedinstvena ulazna točka u aplikaciju.

Dioba koda (code-splitting) (Slika 10.) je postupak dijeljenja paketa aplikacije u manje dijelove koje zahtijeva svaka ulazna točka. Cilj je poboljšati početno vrijeme učitavanja aplikacije učitavanjem samo koda potrebnog za pokretanje te stranice.



Slika 10. Next.js, Code-splitting

(https://nextjs.org/learn/foundations/how-nextjs-works/code-splitting)

Next.js obrađuje većinu ovih transformacija koda i temeljne infrastrukture kako bi aplikaciji olakšao puštanje u proizvodnju.

Što se tiče stranica u aplikaciji, cijela **struktura usmjeravanja** temelji se na datotečnom sustavu. Kad god se stavi JavaScript datoteka u **pages/** datoteku, to je automatski ruta, konfiguracija nije potrebna (pa će npr. pages/ *kontakt* contact.js postati *example.com/contact*).

Uz ovo se mogu uključiti **dinamičke rute** (nazivi varijabli) i napraviti plitko usmjeravanje (što znači da se može promijeniti URL bez ponovnog pozivanja metoda dohvaćanja podataka).

2.7. Mantine

Mantine **[11]** je biblioteka React komponenti. Komponenta je jedan element ili grupa elemenata koji zajedno stvaraju entitet. Izlaz komponente stvorene unutar okvira uvijek je HTML, CSS i JavaScript na kraju.

Biblioteka komponenti je skup komponenti koje se mogu ponovno koristiti. To može biti mapa unutar projekta s uobičajenim komponentama koje se koriste u cijeloj aplikaciji.

Mantine omogućuje korištenje predefiniranih komponenti instaliranjem NPM paketa i dodatnih paketa koji omogućuje dodatne funkcionalnosti poput upravljanja formama i ostalim komponentama.

Svaka komponenta ima dodatne parametre koji definiraju izgled i ponašanje komponente, pa tako komponenta za button ima "*variant*" parametar koji mijenja izgled komponente.

Slika 11. Mantine, Varijante gumba

Osim toga, Mantine omogućuje postavljanje tema koje omogućuju lagane izmjene izgleda aplikacije u potpunosti ovisno o korisničkom odabiru i uređivanje komponenata po vlastitom ukusu u samom JavaScript kodu jer koristi Emotion kao podlogu za stiliziranje komponenti.

Emotion je biblioteka dizajnirana za pisanje css stilova s JavaScriptom.

Dodatne funkcionalnosti koje Mantine omogućuje su responzivni dizajn, upravljanje lokalnom memorijom komponente, korisnički unos podataka, upload datoteka, notifikacije, bogati uređivač teksta i još mnogo ostalih.

2.8. SWR

SWR **[12]** je React biblioteka za dohvaćanje podataka temeljena na SWR strategiji

SWR (**State While Revalidate**) je strategija za prvo vraćanje podataka iz predmemorije (zastarjeli), zatim slanje zahtjeva za dohvaćanje (ponovna provjera valjanosti) i konačno dobivanje ažuriranih podataka.

Prednosti SWR-a:

- **Ponovna provjera fokusa** - kada se vrati fokus na stranicu ili se prelazi s jedne kartice na drugu u pregledniku, SWR automatski ponovno provjerava valjanost podataka.
- **Brza navigacija** - automatski ponovno provjerava valjanost podataka iz izvora čim se podaci renderiraju iz predmemorije.
- **Ponovno dohvaćanje u intervalima** - SWR će dati opciju automatskog dohvaćanja podataka, pri čemu će se prethodno dohvaćanje odvijati samo za komponentu na zaslonu.

- **Lokalna mutacija** - primjena promjena na podatke lokalno, tj. uvijek ažuriranje na najnovije podatke.
- **Ovisno dohvaćanje** - SWR omogućuje dohvaćanje podataka koji ovise o drugim podacima. Osigurava najveći mogući paralelizam (izbjegavanje slapova), kao i serijsko dohvaćanje kada je dio dinamičkih podataka potreban za dohvaćanje sljedećeg podatka.
- **Skalabilan** - SWR se iznimno dobro skalira jer zahtijeva vrlo malo truda da se napišu aplikacije koje automatski i na kraju konvergiraju na najnovije udaljene podatke.

2.9. GitHub i Git

Na visokoj razini, GitHub **[13]** je web stranica i servis temeljen na oblaku koji programerima pomaže u pohranjivanju i upravljanju svojim kodom, kao i praćenju i kontroli promjena koda.

Git služi kao kontrola verzija i pomaže programerima u praćenju i upravljanju promjenama koda softverskog projekta. Kako softverski projekt raste, kontrola verzija postaje bitnija

Izvorni kod web aplikacije Biblio je otvoren i pohranjen na GitHub repozitoriju u oblaku (https://github.com/MVidicek/biblio-bookshelf) gdje svatko može vidjeti i koristiti kod te doprinijeti razvoju aplikacije kreirajući Pull Request.

Zahtjevom za povlačenjem (Pull Request) programer predlaže da se promjene spoje s glavnim kodom.

2.10. Netlify

Netlify **[14]** je platforma za web razvoj čija je namjena povećati produktivnost. Platforma pomaže programerima u izradi, testiranju i postavljanju web stranica.
Objedinjavanjem modernih odvojenih web elemenata od lokalnih razvojnih procesa do napredne logike.

Netlify radi tako da se poveže s GitHub repozitorijem kako bi izvukao izvorni kod, a zatim pokreće proces izrade (build) kako bi unaprijed prikazao sve stranice u statičkom HTML-u.

Ukratko, Netlify stvara vlastitu vrstu repozitorija, a zatim izvršava i distribuira sadržaj preko širokog CDN-a (Content Delivery Network) kako bi mogao isporučiti unaprijed izrađene statične web stranice posjetiteljima.

Netlifyja odabire najbolji CDN i distribuira sadržaj, koji rezultira unaprijed izgrađenim web stranicama koje se učitavaju brže nego na tradicionalnim mrežama za hosting.

Umjesto učitavanje stranice svaki put kada posjetitelj ode na stranicu, posjetitelj dobiva unaprijed učitanu verziju izravno s najbližeg geografskog poslužitelja, bitno smanjujući vrijeme učitavanja.

To znači da se dobiveni HTML, CSS i JS zatim implementiraju i distribuiraju u veliki broj mreža za isporuku sadržaja. Kada posjetitelj pokuša pristupiti web stranici, Netlify automatski odabire najbliži podatkovni centar i poslužuje statičke datoteke.

Netlify poslužuje statične web stranice i osigurava bolju sigurnost, performanse, brzinu i skalabilnost na učinkovitiji način.

2.11. Dodatne Biblioteke

Dodatni NPM paketi korišteni u izradi web aplikacije Biblio:

- **Lottie-react (https://www.npmjs.com/package/lottie-react)** - NPM paket koji omogućuje korištenje Lottie json animacija u React komponentama,
- **React-flags-select (https://www.npmjs.com/package/react-flags-select)** - NPM paket koji pruža komponentu za odabir jezika bilo koje države,
- **Date-fns(https://www.npmjs.com/package/date-fns)** - NPM paket za formatiranje datuma,
- **@radix-ui/react-icons (https://www.npmjs.com/package/@radix-ui/react-icons)** - Pruža ikone u obliku React komponentu.

3. Web Aplikacija Biblio

U ovome poglavlju opisan je proces izrade web aplikacije Biblio.

3.1. Proces razvoja web aplikacije Biblio

Kroz nastavak ovog diplomskog rada će se opisati tehnologije korištene za izradu web aplikacije i sam postupak izrade web aplikacije Biblio.

Faza **dizajna** će se preskočiti jer osobno smatram da je moguće dizajnirati aplikaciju postepeno tijekom razvoja u slučaju kada web aplikacija nije previše kompleksna.
Osim toga Mantine omogućuje korištenje već dizajniranih komponenta što olakšava prilagodbu dizajna web aplikacije tijekom razvoja.

Za **backend** će se koristiti Google Firebase koji omogućuje pristup API pristupnim točkama za autentifikaciju korisnika i pohranu podataka u noSQL bazu podataka. Firebase nudi više usluga, no za ovaj projekt će biti potrebni samo moduli za autentifikaciju (Authentification) i pohranu podataka (Firestore Database). Dakle, mora se napraviti Firebase projekt i implementirati isti u kod za aplikaciju uz pomoć potrebnih biblioteka kako bi se povezali s API točkama.

Što se tiče **frontend** razvoja web aplikacije, koristit će se okvir Next.js koji je baziran na Reactu. React je JavaScript biblioteka za izgradnju interaktivnih korisničkih sučelja.
Pod korisničkim sučeljima mislimo na elemente koje korisnici vide i s kojima komuniciraju na ekranu. React pruža korisne funkcije za izgradnju korisničkog sučelja, ali prepušta razvojnom programeru gdje će koristiti te funkcije u svojoj aplikaciji.

Next.js obrađuje alate i konfiguraciju potrebne za React, te pruža dodatnu strukturu, značajke i optimizacije za aplikaciju.

Izvorni kod aplikacije bit će pohranjen na GitHub-u. GitHub je usluga internetskog hostinga za razvoj softvera i kontrolu verzija softvera pomoću Gita. Omogućuje distribuiranu kontrolu verzija Git plus kontrolu pristupa, praćenje bugova, zahtjeve softverskih značajki, upravljanje zadacima, kontinuiranu integraciju i wiki za svaki projekt.

Naposljetku, web aplikacija mora biti poslužena klijentu s nekog servera. Za to će se koristiti Netlify. Netlify je tvrtka koja nudi hosting i pozadinske usluge bez poslužitelja (serverless) za web aplikacije i statične web stranice.

Tvrtka pruža hosting za web stranice čije su izvorne datoteke pohranjene u sustavu kontrole verzija Git i zatim generirane u datoteke statičnog web sadržaja koje se poslužuju putem mreže za isporuku sadržaja (CDN).

3.2. Priprema radnog okruženja

Na početku je potrebno instalirati i pripremiti neke tehnologije potrebne za izradu aplikacije poput alata za pisanje koda i Node.js kako bi mogli dohvatiti i pokrenuti potrebne NPM pakete.

Za uređivanje koda korišten je **VS Code (https://code.visualstudio.com)**. Visual Studio Code je besplatni uređivač koda koji pomaže programeru u pisanju koda i otklanjanju pogrešaka. Moguće ga je instalirati na bilo koji operacijski sustav.

Potrebno je instalirati **Node.js (https://nodejs.org/en/) [15]** koji omogućuje pokretanje JavaScript koda na poslužitelju i omogućuje nam korištenje **NPM**-a. **Node Package Manager** se instalira automatski uz Node.js.

Nakon toga možemo koristiti NPM CLI (Command Line Interface) **[16]** komande u windows powershell terminalu koje nam omogućuju da dohvatimo NPM pakete i dodatne funkcionalnosti.

```
npm install -g npm // Instalacija zadnje verzije NPM-a
npm install npm-package // Instalacija paketa
```

Kreiran je **GitHub** repozitorij pod nazivom "*biblio-bookshelf*" koji će sadržavati izvorni kod aplikacije u oblaku.

Kako bi mogli sinkronizirati promijene na lokalnom repozitoriju potrebno je instalirati **Git (https://git-scm.com)**.

Git je distribuirani sustav kontrole verzija i u osnovi uzima sliku izgleda datoteka u nekom trenutku te pohranjuje referencu na tu snimku u obliku commit-a.

Omogućuje nam povezivanje s remote repozitorijem i pohranjivanje izmjena u kodu uz pomoć sljedećih CLI komandi:

```
// Inicijalno povezivanje
git remote add origin [url] // Povezivanje sa repozitorijem
git pull origin main // "Povlači" se repozitorij lokalno sa grane "main"

// Za svaku promjenu koju želimo pohraniti
git add -A // Dodavanje svih izmjena u snapshot/commit
git commit -m 'text' // Kreira se commit sa opisom promjena
git push origin main // "Guraju" se promijene na "main" granu
```

3.3. Kreiranje Next.js aplikacije

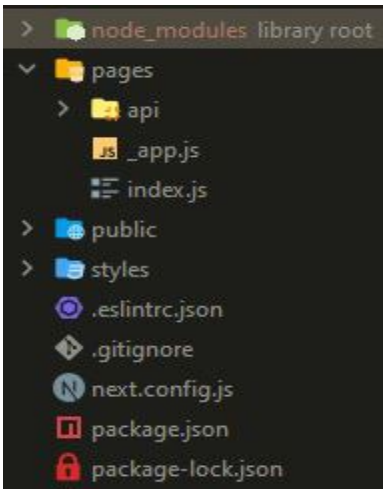
Aplikaciju kreiramo koristeći *"**create-next-app**"* komandu, koja automatski postavlja projekt. Za izradu projekta se pokreće sljedeća komanda:

```
npx create-next-app@latest biblio-bookshelf
```

NPX je kratica za Node Package eXecute. U svojoj srži, npx je npm pokretač paketa koji se koristi za izvršavanje paketa bez potrebe za njihovom globalnom instalacijom.

Pokretanjem paketa kreira se direktorij pod nazivom "biblio-bookshelf" koji sadržava sve resurse potrebne za pokretanje i razvoj Next.js aplikacije. Zatim ulazimo u direktorij i pokrećemo VS Code na sljedeći način:

```
cd biblio-bookshelf
. code // Pokreće instancu VS Code
```



Slika 12. Početna struktura aplikacije

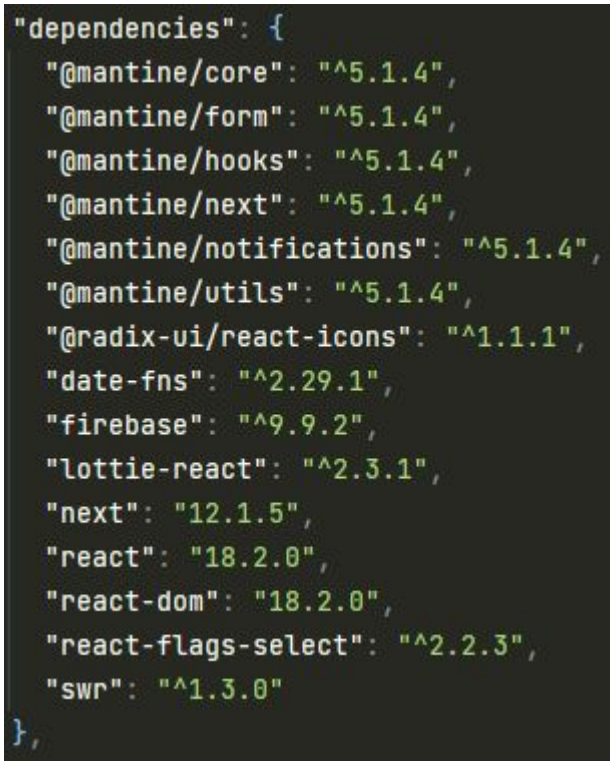
Stvoreni direktorij inicijalno sadržava određene datoteke i direktorije uključujući (Slika 12.):

- **node_modules** - Svi instalirani NPM paketi, ne "gura" se na remote repozitorij jer sadržava veliku količinu podataka. Svatko može po potrebi instalirati potrebne pakete sa naredbom "*npm install*" nakon što lokalno klonira repozitorij.
- **pages** - JavaScript datoteke koje definiraju pojedine stranice (rute) web aplikacije. Inicijalno je stvorena JavaScript datoteka ***_app.js*** koja sadržava cijelu aplikaciju i prikazuje se na svakoj stranici tj. ostatak aplikacije se izvršava unutar nje.
- **public** - Javne datoteke poput favicon ikona, slika i grafika.
- **styles** - CSS datoteke.

Od ostalih datoteka bitnije su **.gitignore** (Govori Git-u koje datoteke se ignoriraju prilikom commit-a) i **package.json** (slika 13.) koji sadržava sve informacije o NPM paketima i skriptama za pokretanje i izgradnju aplikacije.

Nakon dovršetka instalacije može se pokrenuti skripta za otvaranje development servera korištenjem naredbe "***npm run dev"*** na lokalnom portu 3000 (http://localhost:3000). Automatski se osvježava svaki puta kada se detektira promjena u kodu prilikom spremanja.

Sljedeći korak je instalacija NPM paketa korištenjem naredbe "***npm install***".



Slika 13. Lista instaliranih NPM paketa u package.json datoteci

3.4. Google Firebase konfiguracija

Prije nego što aplikacija može koristiti Firebase usluge, treba izraditi Firebase projekt i registrirati svoju aplikaciju s tim projektom **[15]**. Kada se registrira aplikacija u Firebase konzoli, dobije se Firebase konfiguracijski objekt koji će se koristiti za povezivanje aplikacije s resursima Firebase projekta.

Osim toga, potrebno je instalirati SKD paket i inicijalizirati Firebase i podpakete usluga koje želimo koristiti u svojoj aplikaciji (U ovome slučaju to su Firestore i Authentication).

```
import { initializeApp } from 'firebase/app';

const firebaseConfig = {
  //... Firebase konfiguracija za projekt ide ovdje
};

const app = initializeApp(firebaseConfig);

// Inicijaliziramo usluge koje želimo koristiti i izvozimo ih kako bi ih
// mogli koristiti u drugim datotekama
export const db = getFirestore(app); // Firestore baza podataka
export const auth = getAuth(app); // Usluga za autentifikaciju
```

Firebase objekt "*app*" je nalik spremniku koji pohranjuje zajedničku konfiguraciju i dijeli autentifikaciju preko Firebase usluga. Nakon što se inicijalizira Firebase App objekt u svom kodu, mogu se početi koristiti Firebase usluge.

3.5. Planiranje rasporeda stranica

Slika 14. pages/dashboard.js, Glavna stranica

Mnoge komponente često se ponovno koriste između stranica pa tako možemo stvoriti "**Layout**" komponentu koja će se prikazati na svakoj stranici.
Pošto u ovome slučaju stranice imaju drugačiji raspored (layout), dodaje se svojstvo **getLayout**, što omogućuje definiranje izgleda za svaku stranicu posebno.

```
// layouts/Layout.js
export const getLayout = (page) => <Layout>{page}</Layout>;

// pages/index.js

Page.getLayout = function getLayout;

// pages/_app.js
export default function MyApp({ Component, pageProps }) {
  // Koristimo Layout definiran u stranici, ako postoji
  const getLayout = Component.getLayout || ((page) => page)
  return getLayout(<Component {...pageProps} />)
}
```

Layout komponenta za Dashboard (Slika 14.) sadrži **Header, Navigation i Aside** (Sidebar) pod komponente (Svaka od njih sadrži svoje unutarnje komponente) te unutar sebe prikazuje dijete tj. komponentu koja se prikazuje unutar Layout-a (**Main Body**).

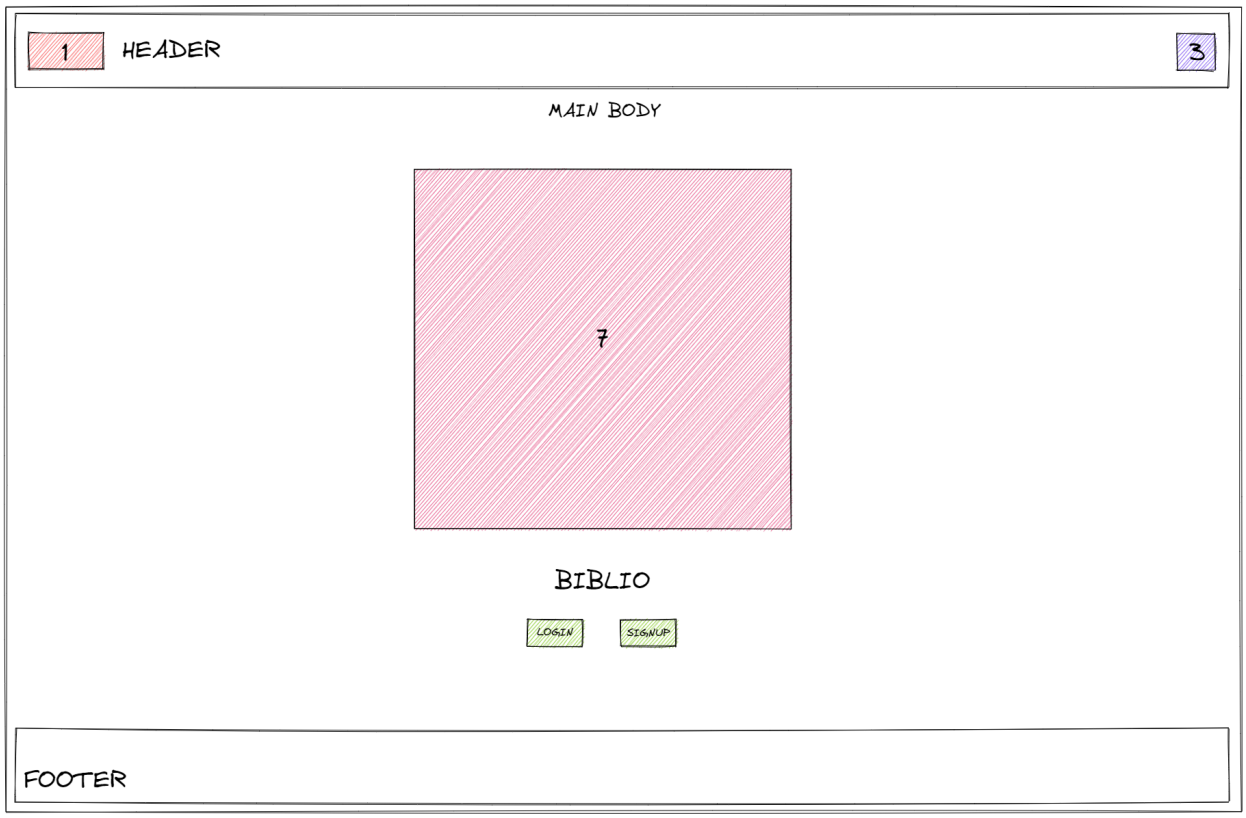
HEADER
1 - Logo
2 - Header Title / Search Form - Ovisno o navigaciji korisnika prikazuje se tekst ("Bookmarks", "Home", "Reading", itd.), ukoliko je prikazana "Discover" komponenta pojaviti će se forma za pretraživanje
3 - Theme Switcher - Komponenta koja omogućuje promjenu teme (Light/Dark)

NAVIGATION
4 - Main Links - Gumbi za navigaciju (Home, Discover, Reading, Finished, Bookmarks i Account Settings)

SIDEBAR
5 - Search Settings - Postavke pretraživanja (parametri upita koji se šalju na API endpoint) i odabir API-a (Google Books, Open Library)

MAIN
6 - Ovdje će se ovisno o pristisku na navigacijski gumb prikazati određena glavna komponenta

Osim glavne stranice (***pages/dashboard.js***) postoji i početna stranica za autentifikaciju korisnika (***pages/index.js***) koja ima svoj zasebni Layout (slika 15.) bez određenih dijelova i komponenti.



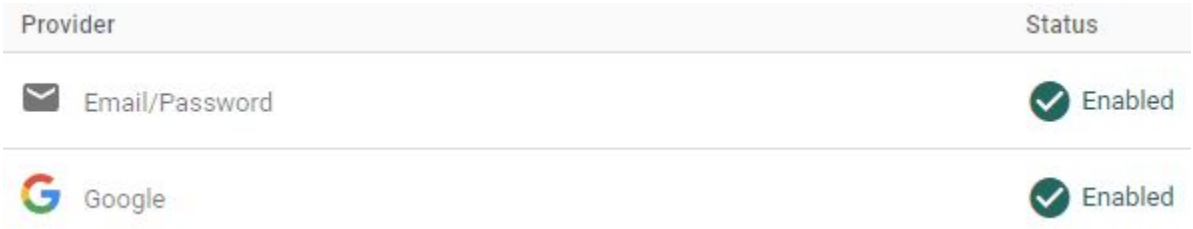
Slika 15. pages/index.js, Login

Nisu potrebni “Navigation” i “Sidebar” dijelovi glavne aplikacije dok header ne sadrži naslove i formu za pretragu. Uključen je Footer za neke dodatne informacije o stranici.

Glavni dio Layout komponente prikazuje sadržaj index.js datoteke koja uključuje komponentu za prikaz Lottie animacije (**7.**), naslov aplikacije te gumbe za prijavu korisnika i kreiranje novog računa koji otvaraju modul za unos podataka.

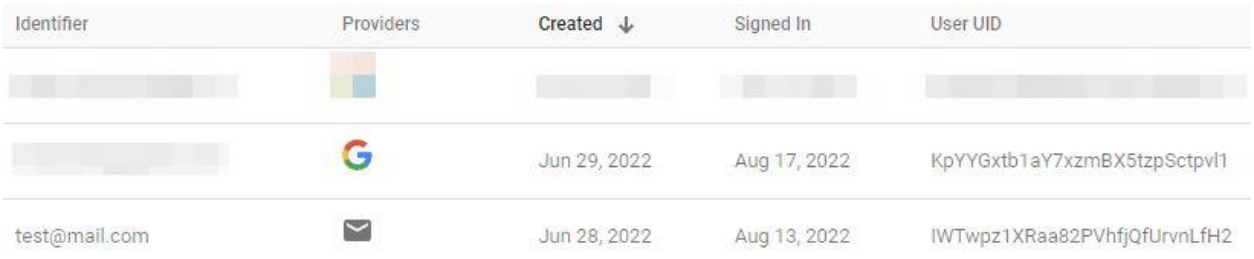
3.6. Autentifikacija

U konzoli za Firebase projekt potrebno je uključiti željene opcije za kreiranje korisnika (Slika 16).



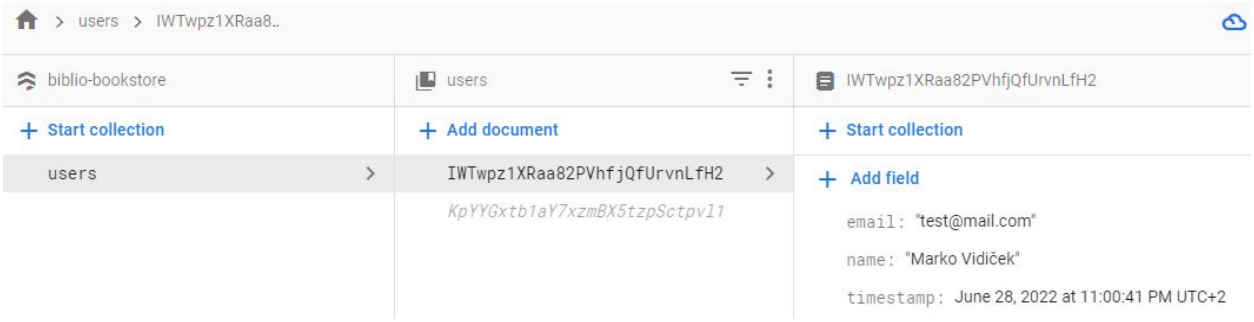
Slika 16. Pružatelji za autentifikaciju

Korisnik može koristiti navedene opcije za pristup aplikaciji i kreiranje novog računa.
Za svakog korisnika stvara se zapis u “users” tablicu usluge za autentifikaciju.



Slika 17.Firebase, popis korisnika

Osim toga, svaki korisnik (Slika 17.) je povezan sa dokumentom u Firestore bazi podataka koji nosi naziv identifikatora korisnika (**User UID**) i sadrži dodatne podatke koje korisnik određuje prilikom kreiranja računa (npr. username) (Slika 18.)



Slika 18. Firestore, "users" dokument

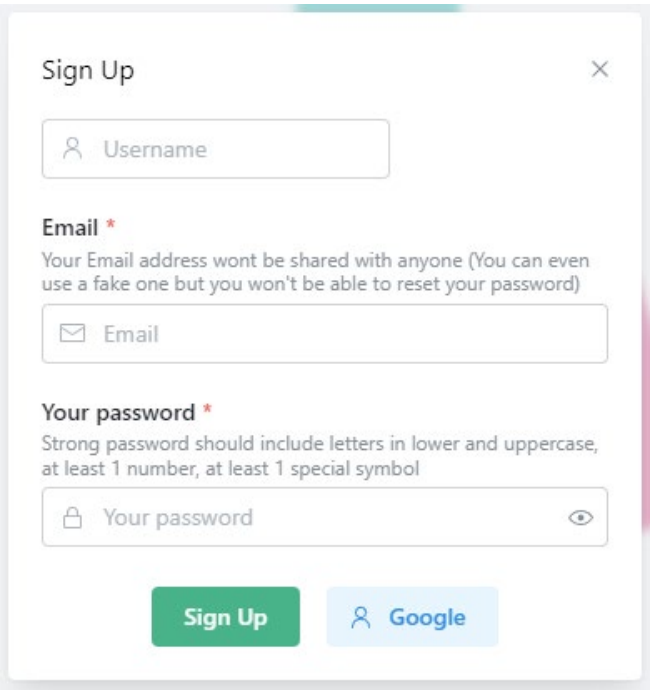
Kako bi korisnicima bilo omogućeno upravljanje samo onim podatcima koji su njihovi, postavljena su sljedeća pravila za noSQL bazu podataka.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    // Users
    match /users/{user} {
      allow read;
      allow create;
      allow update: if request.auth.uid == user }}}
```

Nakon postavljanja usluge unutar Firebase konzole, sada je moguće napraviti komponente unutar aplikacije koje će koristiti Firebase SDK paket kako bi mogle komunicirati sa uslugama za autentifikaciju i bazu podataka te im slati informacije o stvaranju novog korisnika ili prijavu već postojećeg, a kao povrat će dobiti informaciju o autentifikaciji korisnika i kreirati sesiju.

Svaki put kada se korisnik prijavi, korisničke informacije šalju se Authentication usluzi i razmjenjuju se za Firebase ID token (JWT) i token za osvježavanje sesije.

3.6.1. Kreiranje Korisnika



Slika 19. SignUp komponenta

SignUp komponenta (Slika 19.) sadrži Mantine forme za unos korisničkih podataka koji će se koristiti za stvaranje novog korisnika i gumb koji poziva “handleSubmit” funkciju.

```
const form = useForm({
    initialValues: { name: "", email: "" },
    validate: {
      email: (value) => (/^\S+@\S+$/.test(value) ? null : "Invalid email"),
      name: (value) =>
        value.length < 2 ? "Name must have at least 2 letters" : null,
    },});
```

Ova forma prima vrijednosti za email i username te provjerava je li unos ispravan.

```
<TextInput {...form.getInputProps("email")} required … />
```

Mantine “TextInput” komponenta automatski osvježava vrijednosti unutar forme prilikom izmjena u obrascu za unos određenog parametra.

Pritiskom na gumb “Sign Up” izvršava se “handleSubmit” funkcija gdje se kreira novi korisnik uz pomoć firebase SDK metoda i preusmjerava se na Dashboard aplikacije.

```
 const handleSubmit = async (values) => {

    // “values” sadrži sve podatke iz forme
    // Destrukturiraju se email i name varijable
    const { name, email } = values;

    // Unutar try-catch izjave asinkrono se pozivaju metode iz
    firebase/auth paketa za kreiranje korisnika
    try {
      const { user } = await createUserWithEmailAndPassword(
        auth,
        email,
        password
      );

      // Kopiraju se podaci iz forme i dodaje im se timestamp datuma
      const formData = { ...values };
      formData.timestamp = serverTimestamp();

// Poziva se setDoc metoda iz firebase/firestore paketa koja stvara novi dokument
na referenci koju određuje “doc” metoda uz pomoć parametra koji definiraju gdje će
se dokument nalaziti i kako će se zvati (db -> baza  podataka, “users” kolekcija
dokumenata, user UID kao naziv novog dokumenta)

      await setDoc(doc(db, "users", user.uid), formData);

      // Preusmjeravanje na Dashboard
      router.push("/dashboard");
    } catch (error) {...error handling});}};
```

3.6.2. Prijava Korisnika

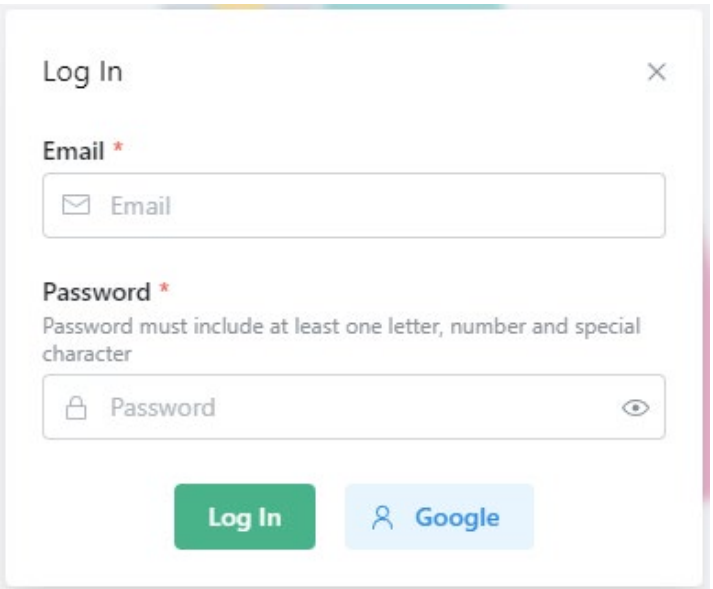
Prijava je moguća preko google OAuth pružatelja ili upisivanjem odgovarajuće email/password kombinacije u Login komponenti (Slika 20.).

Postoji forma koja služi za pohranu email-a i lozinke u obliku stringa te Mantine input komponente i funkcija koja se izvršava klikom na gumb za prijavu.
Samo se ovaj puta poziva **signInWithEmailAndPassword** metoda SDK-a.

```
const handleSubmit = async (values) => {
    const { email, password } = values;
```

```
    try {
      const userCredential = await signInWithEmailAndPassword(
        auth,
        email,
        password
      );

      if (userCredential.user) {
        router.push("/dashboard");}
    } catch (error) {...error handling});}};
```



Slika 20. LogIn komponenta

3.6.3. OAuth

Korisnici se mogu prijaviti koristeći svoj Google račun, Google gumb je posebna komponenta koja sadrži kod koji izvršava autentifikaciju korisnika preko google OAuth usluge. Unutar asinkrone “onGoogleClick” funkcije izvršava se kod koji uključuje sljedeće metode:

```
const provider = new GoogleAuthProvider();
const { user } = await signInWithPopup(auth, provider);
```

Osim toga stvara se referenca na bazu podataka i “snimka” baze unutar komponente kako bi provjerili postoji li dokument ili ne. Ukoliko ne postoji, stvara se dokument za novog korisnika.

```
const docRef = doc(db, "users", user.uid);
const docSnap = await getDoc(docRef);
if (!docSnap.exists) { setDoc ...
```

3.7. Layout Komponenta

```
import ...
function Layout({ children }) {
  const [page, setPage] = useState("home");

  return (
    <AppShell
      navbar={<LayoutNavbar page={page} setPage={setPage} />}
      aside={<LayoutAside />}
      header={<LayoutHeader page={page} />}>
       // Ovdje se nalazi glavna komponenta
      <Center>{React.cloneElement(children, { page, setPage })}</Center>
    </AppShell>
  );}
```

React ima funkcionalnost postavljanja i modificiranja unutarneg stanja (memorije) komponente pomoću "**useState**" funkcije. Tako možemo postaviti "**page**" string unutar Layout komponente na neku vrijednost i proslijediti funkciju "**setPage**", koja služi za izmjenu iste varijable, komponentama unutar Layout-a.

U Reactu se dijeljenje ovih varijabli stanja postiže pomicanjem do najbližeg zajedničkog pretka komponenti koje ga trebaju. To se zove "**lifting state up**". Pa tako sve komponente unutar Layout-a mogu primiti ovu varijablu kao argument i koristiti/modificirati je.

Ovisno o "page", varijabli se vraća određena komponenta unutar pages/dashboard.js. U ovom slučaju varijabla se naziva "pages" iako nema veze sa promjenom stranice na kojoj se nalazimo već samo mijenja komponentu koja se prikazuje unutar **AppShell** komponente.

```
export default function Dashboard({ page, setPage }) {
  if (page === "profile") return <Profile />;
  if (page === "discover") return <Discover />; ...
```

React komponente se regeneriraju svaki puta kada se promijeni njihov ulazni argument ili untarnje stanje pa tako prilikom promjene "page" varijable klikom na navigacijski gumb, dashboard.js se opet pokreće i vraća drugu komponentu.

AppShell je Mantine komponenta koja se može koristiti za stvaranje zajedničkog uzorka rasporeda zaglavlja - navigacijske trake - podnožja - aside - sadržaja.

AppShell prima argumente čiji naziv određuje koja komponenta se koristi za prikaz određenog dijela Layout-a (navbar={}, header={} …)

3.7.1. Navbar

```
export default function LayoutNavbar({ page, setPage }) {
  return (
    <Navbar>
        <MainLinks setPage={setPage} />
        <User setPage={setPage} />
    </Navbar> );}
```

Navigacijska komponenta sadrži **MainLinks** i **User** komponente.
MainLinks komponenta generira navigacijske elemente u obliku Mantine “UnstyledButton” komponenti. Definiraju se podaci za svaki element unutar “data” polja kao objekti koji sadrže vrijednosti za ikonu, boju i oznaku. Zatim se u glavnoj izlaznoj komponenti mapira kroz polje objekata i generira specifična MainLink komponenta ovisno o ulaznim podacima.

```
function MainLink({ icon, color, label, setPage }) {
  return (
    <UnstyledButton
      onClick={() => setPage(label.toLowerCase())} >
      <Group>
        <ThemeIcon color={color} variant="light">
          {icon}
        </ThemeIcon>
        <Text size="sm">{label}</Text>
      </Group>
    </UnstyledButton> );}
```

```
// Podaci za pojedini link
const data = [
  { icon: <HomeIcon />, color: "gray", label: "Home" },
  { icon: <MagnifyingGlassIcon />, color: "gray", label: "Discover" }, ... ];

// Glavna komponenta mapira kroz “data” polje i kreira za svaki objekt unutar tog
polje odgovaraju komponentu
export function MainLinks({ setPage }) {
  const links = data.map((link) => (
    <MainLink setPage={setPage} {...link} key={link.label} />
  ));
  return <div>{links}</div>;}
```

Svakoj MainLink komponenti se prosljeđuje “*setPage*” koji se poziva klikom na gumb i postavlja “*page*” varijablu stanja ovisno o oznaci (*label*). Kao reakcija na to, regenerira se stranica i montira druga komponenta unutar dashboard.js
Users komponenta sadrži grupu Mantine komponenti uključujući gumb koji postavlja “page” na “profile” (*onClick={() => setPage("profile")}*)

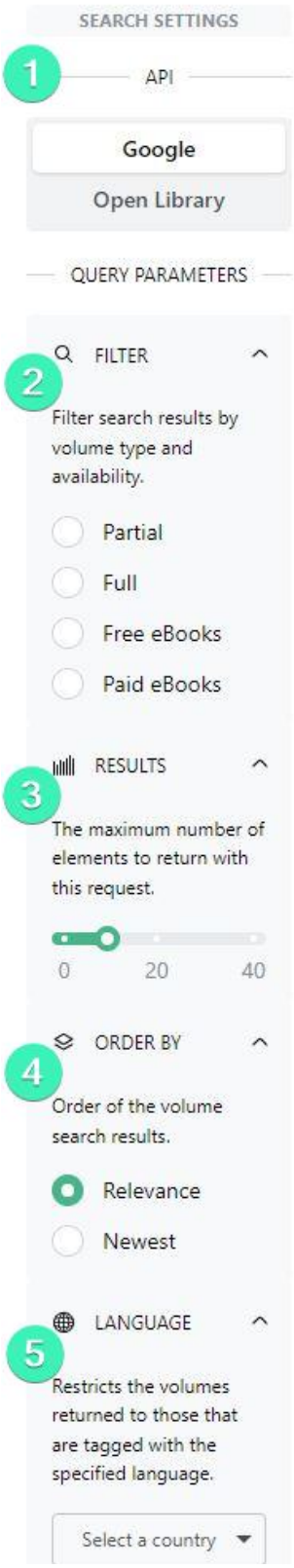
3.7.2. Aside

"Aside" komponenta (Slika 21.) sadrži Input komponente za konfiguraciju postavka pretraživanja u obliku parametra koji će se slati uz upit na API endpoint. Trenutno ne pruža nikakve funkcionalnosti već samo služi kao placeholder.

Svi parametri upita se nalaze unutar **Accordion** (https://mantine.dev/core/accordion/) komponente koja omogućuje otvaranje/zatvaranje pojedinog parametra.

Na slici 22 prikazana je sidebar komponenta koja sadrži;
(1) - **SegmentedControl** (https://mantine.dev/core/segmented-control/) Mantine komponenta koja služi za odabir API endpoint-a
(2) - **Radio Button** (https://mantine.dev/core/radio/) komponenta gdje korisnik može filtrirati dobivene rezultate ovisno o količini podataka i eBooks dostupnosti.
(3) - **Slider** (https://mantine.dev/core/slider/) komponenta za limitiranje količine rezultata (40 max)
(4) - **Radio Button** za poredak rezultata po relevantnosti ili datumu
(5) - Odabir jezika uz pomoć **ReactFlagsSelect** komponente iz react-flags-select npm paketa

Ovisno o odabranim parametrima mijenjat će se globalne varijable pretrage koje će SWR funkcija za dohvaćanje podataka primati kao argumente i time dohvaćati određene rezultate sa Google Books API-a.



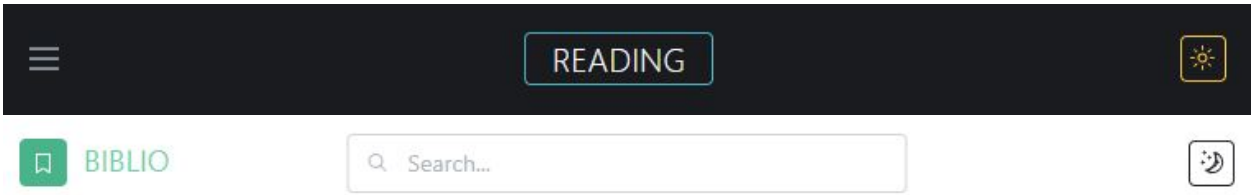
Slika 21. Aside komponenta

3.7.3. Header

Header komponenta (Slika 22.) uključuje Logo (**Logo.jsx**) koji se prikazuje ovisno o širini ekrana. Ako je korisnik na ekranu manjem od specificirane širine, prikazuje se ikona za otvaranje "burger" menia za navigaciju.

Osim toga, u Header komponenti nalaze se komponente za prikaz naslova (**HeaderTitle.jsx**), unos teksta za pretragu (**SearchForm.jsx**) i promjenu teme (**ThemeSwitcher.jsx**).

Ovisno o "page" varijabli prikazuje se određeni naslov koji koristi Mantine tranzicije (https://mantine.dev/core/transition/) prilikom montiranja komponente.



Slika 22. Header komponenta

Kako se ne bi morali prosljeđivati “search” varijablu iz Layout komponente u sve ostale komponente koje ona nasljeđuje i koje koriste “**search**” za dohvaćanje knjiga, može se napraviti custom Hook komponenta tj. pomoćna funkcija za postavljanje globalnog stanja tj. globalne search varijable koja se može postaviti i dohvatiti u bilo kojoj komponenti (**useGlobalState.js**, korištenje prefiksa “use” je konvencija imenovanja top-level funkcija u React okviru koje nazivamo “Hooks” i izvršavaju se samo na najvišem nivou komponenta, a ne u funkcijama ili petljama).

```
import useSWR from "swr";

export default const useGlobalState = (key, initialData) => {
  const { data, mutate } = useSWR(key, () => initialData, {
    revalidateOnFocus: false,
    revalidateOnMount: false,
    revalidateIfStale: false,
  });
  return [
    data ?? initialData,
    (value) =>
      mutate(value, {
        revalidate: false,
      }),];};
```

Kao što vidimo, definira se izlazna funkcija useGlobalState koja kao argumente prima **key** (ključ) po kojem će SWR znati o kojoj se vrijednosti radi u cache memoriji i **initialData** (podatak) koji će se pohraniti uz taj ključ.
Funkcija vraća polje koje sadrži varijablu u memoriji ukoliko postoji ili initialData i funkciju za mutaciju vrijednosti u memoriji.

```
const [search, setSearch] = useGlobalState("search", "");
```

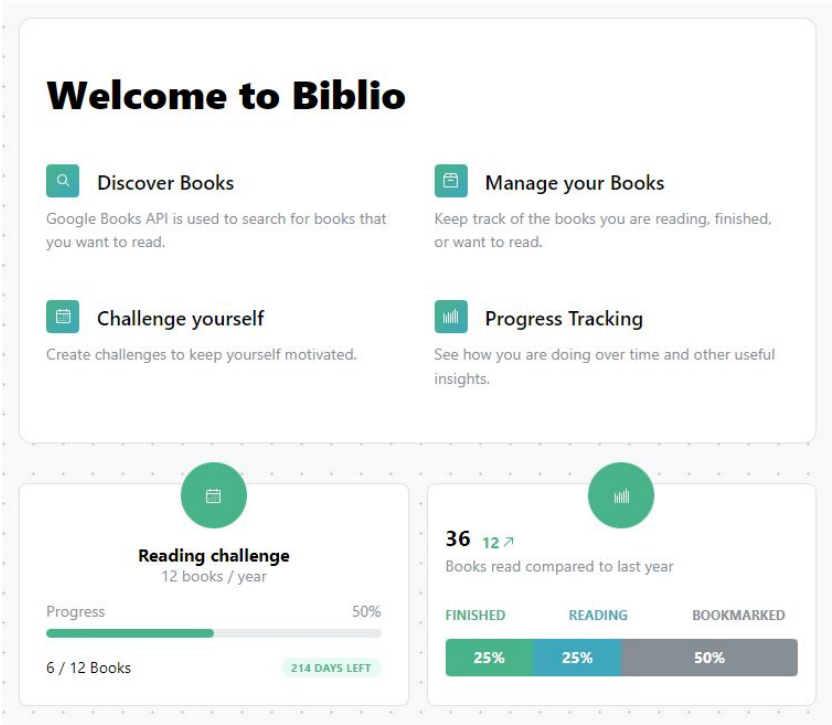
Može se koristiti Input komponenta za promjenu podataka u memoriji koji se dalje mogu koristiti u ostalim komponentama pod istim ključem.
Isključuje se revalidiranje kako bi rezultat ostao pohranjen sve dok korisnik ne ugasi prozor u pregledniku.

3.8. Glavne komponente

Pod glavne komponente uključene su komponente za prikaz glavnih funkcionalnosti aplikacije poput pretraživanja knjiga i tablica kolekcija.
To uključuje: Home.jsx, Discover.jsx, Reading.jsx, Finished.jsx, Bookmarks.jsx i Profile.jsx komponente.

3.8.1. Home.jsx

Home komponenta (Slika 23.) trenutno prikazuje neke osnovne informacije i funkcionalnosti aplikacije. Trenutno sadrži placeholder komponente za izazove i statistiku koji za sada nemaju funkcionalnost no ubuduće će korisnik na ovoj "stranici" imati uvid u statistiku i mogućnost kreiranja izazova za čitanje.



Slika 23. Home.jsx komponenta

3.8.2. Discover.jsx

Moglo bi se reći da je ova komponenta srž aplikacije jer nam omogućuje uvid u Google Books API i pretraživanje knjiga, te dodavanje istih u kolekcije. Cijelokupna "stranica" se sastoji od glavne **Discover.jsx** komponente (Slika 24.) koja generira **BookItem.jsx** komponente u kojoj se nalazi gumb za otvaranje **BookDetailsModal.jsx** komponente.

Kako bi mogli dohvatiti knjige, moramo poslati zahtjev na Google Books API, za to je kreirana posebna "Hook" funkcija uz pomoć SWR tehnologije pod imenom **useFetchBooks.js**

```
import useSWR from "swr";
const fetcher = (...args) => fetch(...args).then((res) => res.json());
export default function useFetchBooks(
  startIndex = 0,
  maxResults = 8,
  searchText = ""
) {
  const { data, error } = useSWR(
`https://www.googleapis.com/books/v1/volumes?q=${searchText}&startIndex=${startInde
x}&maxResults=${maxResults}&fields=items`,
    fetcher);
  return {
    books: data?.items,
    isError: error,
  };}
```

Fetcher je ovdje asinkrona funkcija koja prihvaća ključ SWR-a i vraća podatke. Vraćena vrijednost bit će proslijeđena kao podatak, a izbaci li error, bit će uhvaćena kao pogreška.

Izvodi dohvaćanje podataka u **3 koraka**:
1 - prvo vraća podatke iz predmemorije (zastarjeli),
2 - šalje zahtjev za dohvaćanje (ponovno potvrdi),
3 - vraća ažurirane podatke.

Kao argumente funkciji šaljemo parametre upita koji se ubacuju u “url” za API, a na kraju nam funkcija vraća podatke o knjigama i/ili error.

```
export default function Discover() {
  const [pageIndex, setPageIndex] = useState(1);
  const [loading, setLoading] = useState(true);
  const [searchText] = useGlobalState("search", "");

  const { books, isError } = useFetchBooks(
    (pageIndex - 1) * 10, 10, searchText );

  useEffect(() => {
    if (books) setLoading(false);
  }, [books]);

  if (loading || books === null || books === undefined)
    return ... Loader (Lottie)
  if (isError) ... Error notification

  return (
    <div>
```

```
      <SimpleGrid>
        {books.map((book) => {
            return <BookItem key={book.id} book={book} />;})}
      </SimpleGrid>
      <Pagination
        page={pageIndex}
        onChange={setPageIndex} /> </div>);}
```

Komponenta prvo postavlja unutarnja stanja za paginaciju, učitavanje i tekst za pretraživanje iz SWR cache memorije uz pomoć globalnog konteksta.

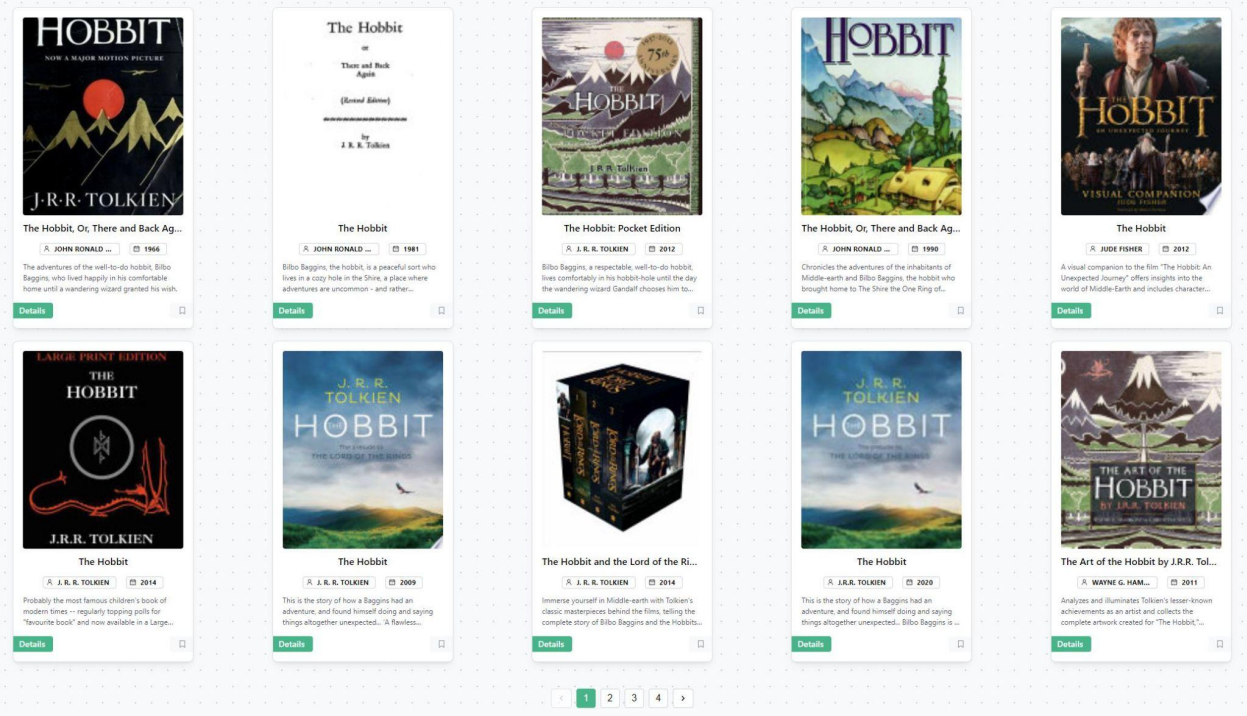
Zatim se poziva **useFetchBooks** sa potrebnim parametrima (Prvi parametar je početni indeks koji ovisi o paginaciji pa se kada primjerice imamo 10 rezultata računa na način: *(pageIndex - 1) * 10*).

useEffect je React funkcija koja omogućuje pokretanja nekih nuspojava unutar komponente prilikom svakom regeneriranja i uz to možemo specificirati o kojim parametrima pokretanje tih funkcionalnosti ovisi unutar zagrada (u ovom slučaju svaki puta kada se promijeni "**books**" varijabla pregledava se postoje li knjige i ako da, stanje **loading** se postavlja na **false**).

Ako knjiga nema, komponenta vraća Loader u obliku animacije, a ako je vraćen error izbacuje notifikaciju s porukom greške.

Ako ima potrebne podatke, vraća se **SimpleGrid** Mantine komponenta mreže koja sadrži **BookItem** komponentu za svaki objekt knjige u polju books kojoj se prosljeđuje objekt knjige s podacima prilikom mapiranja.

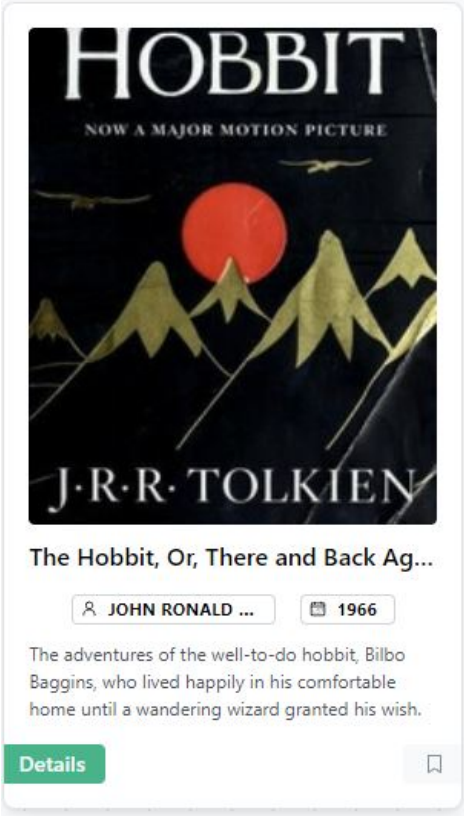
I na kraju Pagination Mantine komponenta koja postavlja stanje stranice i time regenerira komponentu koja poziva pretragu sa novim početnim indeksom.



Slika 24. Discover.jsx komponenta

BookItem komponenta (Slika 25.) je kartica koja sadrži naslovnu sliku knjige i neke osnovne informacije te gumb za dodavanje knjige u “Bookmarks” kolekciju i za otvaranje BookDetailsModal.jsx komponente s detaljima o knjizi.

Za svaku knjigu poziva se **checkBookCollection** funkcija unutar **useEffect**-a kako bi se u stvarnom vremenu mijenjali podaci o tome je li knjiga u bazi podatka i ovisno o tome se mijenja boja gumba kako bi znali da je knjiga već u kolekciji i da će se pritiskom maknuti iz nje.



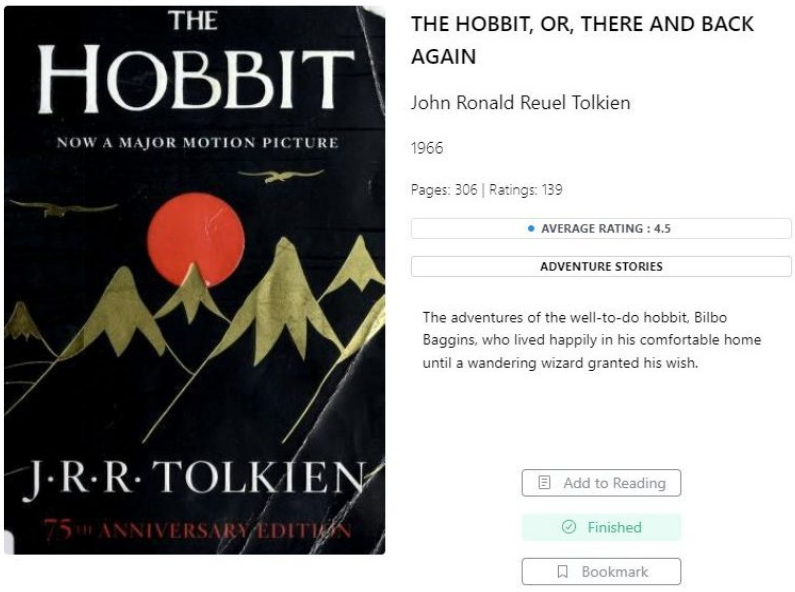
Slika 25. BookItem.jsx komponenta

```
const querySnapshot = await getDocs(
      collection(db, "users", user.uid, colName));
    querySnapshot.forEach((doc) => {
      if (doc.data().bookId === id) {
        setCollection(true);}});
```

Kako bi se pohranile te informacije BookItem ima unutarnje stanje za svaku kolekciju (Reading, Finished, Bookmarks) koje se stavlja na "**true**" vrijednost ako snimka iz baze podataka postoji.

Klikom na bookmark gumb, poziva se ***setBookCollection****(book, isBookmarked, setIsBookmarked, "bookmarked");* - Kao parametre funkcija prima podatke o knjizi koje će zapisati u dokument, varijablu koja funkciji govori nalazi li se već u kolekciji i prema tome vrši brisanje i postavlja tu varijablu na suprotno što je bila, te string koji određuje u kojoj kolekciji će dokument biti pohranjen.

Pritiskom na gumb "Details" učitava se **BookDetailsModal.jsx** komponenta (Slika 26.) kojoj se prosljeđuju podaci o knjizi. Ova komponenta sadrži više formatiranih detalja o pojedinoj knjizi uz gumbe za dodavanje knjige u određene kolekcije koji pozivaju ***setBookCollection*** funkciju s određenim parametrima.



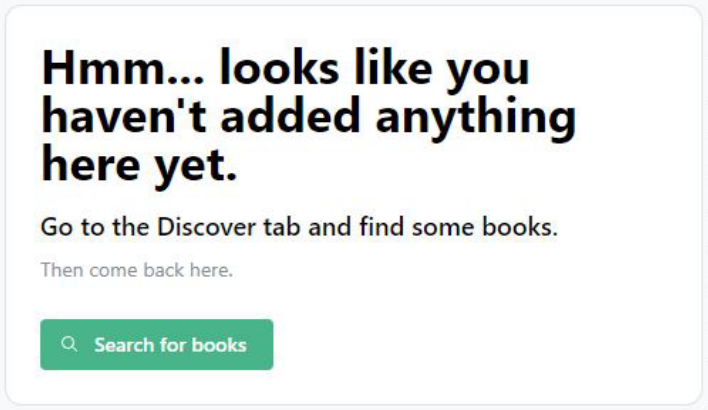
Slika 26. BookDetailsModal.jsx komponenta

3.8.3. Kolekcije (Reading, Finished, Bookmarks)

Ove 3 komponente su većinom iste osim što prikazuju tablice za druge kolekcije knjiga unutar Firestore-a. Trenutno postoje dvije komponente koje se koriste: **BookTable.jsx** i **NoBooksAdded.jsx**

NoBooksAdded.jsx (Slika 27.) je placeholder komponenta kada korisnik nike dodao knjige u kolekciju i navigira korisnika na Discover komponentu.

Glavna komonenta poziva **fetchDocuments**("finished") u useEffect-u s argumentom koji specificira kolekciju knjiga koje se dohvaćaju.

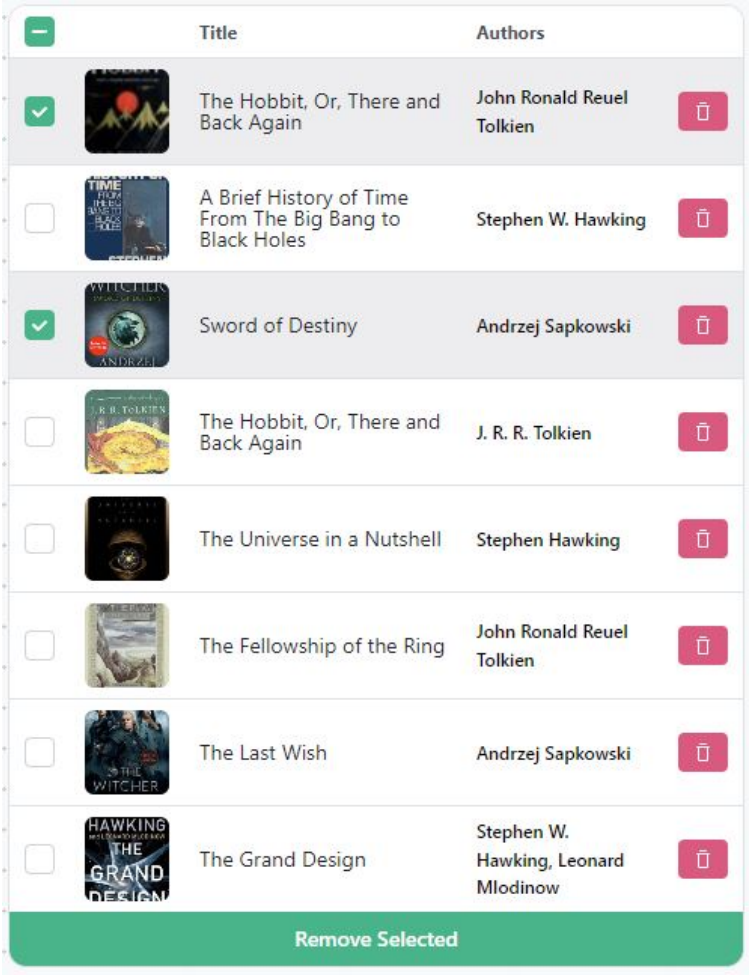


Slika 27. NoBooksAdded.jsx komponenta

Podaci o knjigama uz argumente za postavljanje loading stanja i naziva kolekcije kao string se prosljeđuju **BookTable.jsx** komponenti (slika 28.).

BooksTable mapira kroz podatke i kreira retke tablice za prikaz podataka o knjigama u kolekciji.

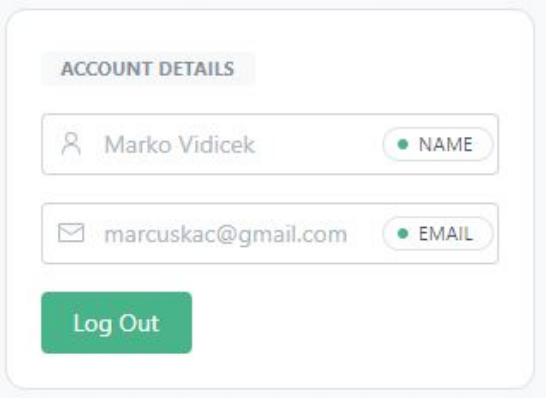
U prvom stupcu se nalazi “checkbox” koji omogućuje označavanje redaka i brisanje više njih iz tablice pritiskom na “Remove Selected” gumb koji poziva **removeDocument** funkciju koja prima polje selektiranih ID-eva i ima kolekcije kao argumente, dok se u zadnjem stupcu nalazi gumb za brisanje pojedinog retka.



Slika 28. BookTable.jsx

3.8.4. Profile

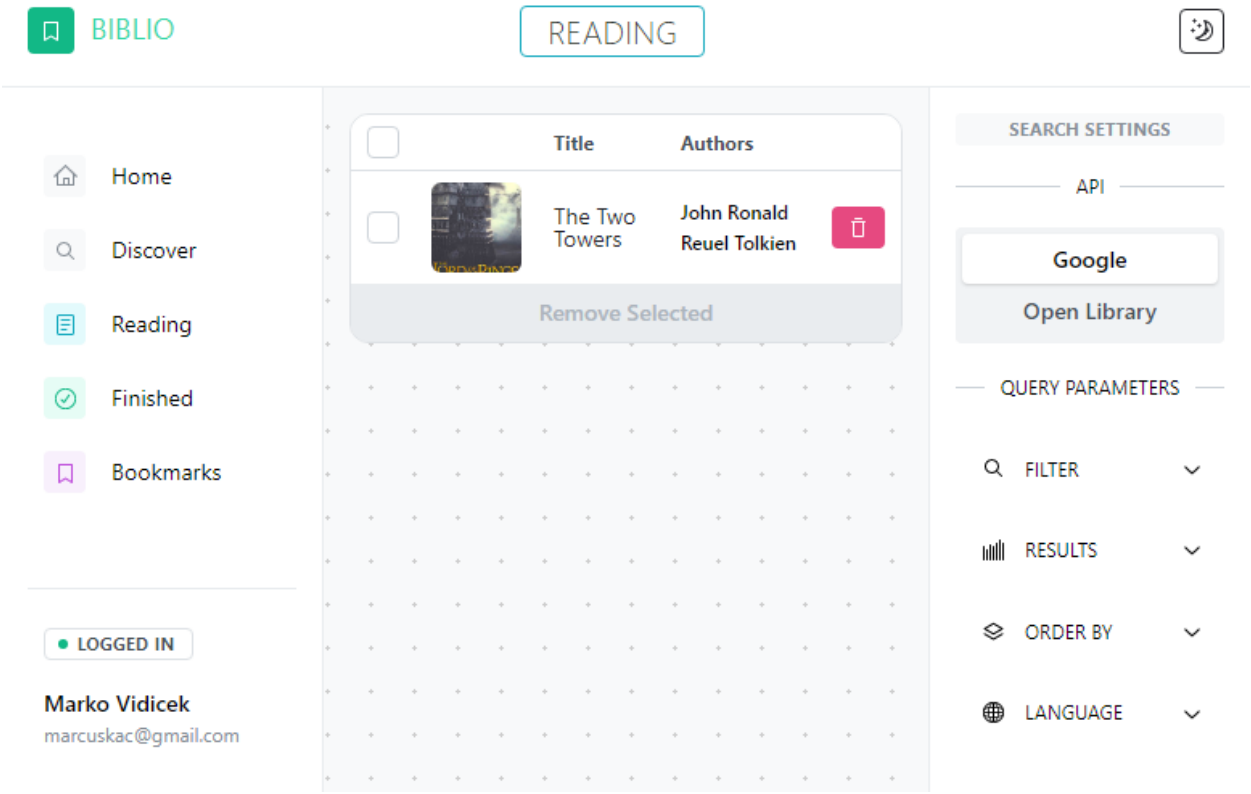
Profile.jsx (Slika 29.) je jednostavna komponenta koja sadrži podatke o korisničkom računu poput imena i e-mail adrese te gumb za izlaz iz aplikacije i zatvaranje sesije s korisnikom. Ubuduće će korisnik imati mogućnost ovdje promijeniti podatke o računu.



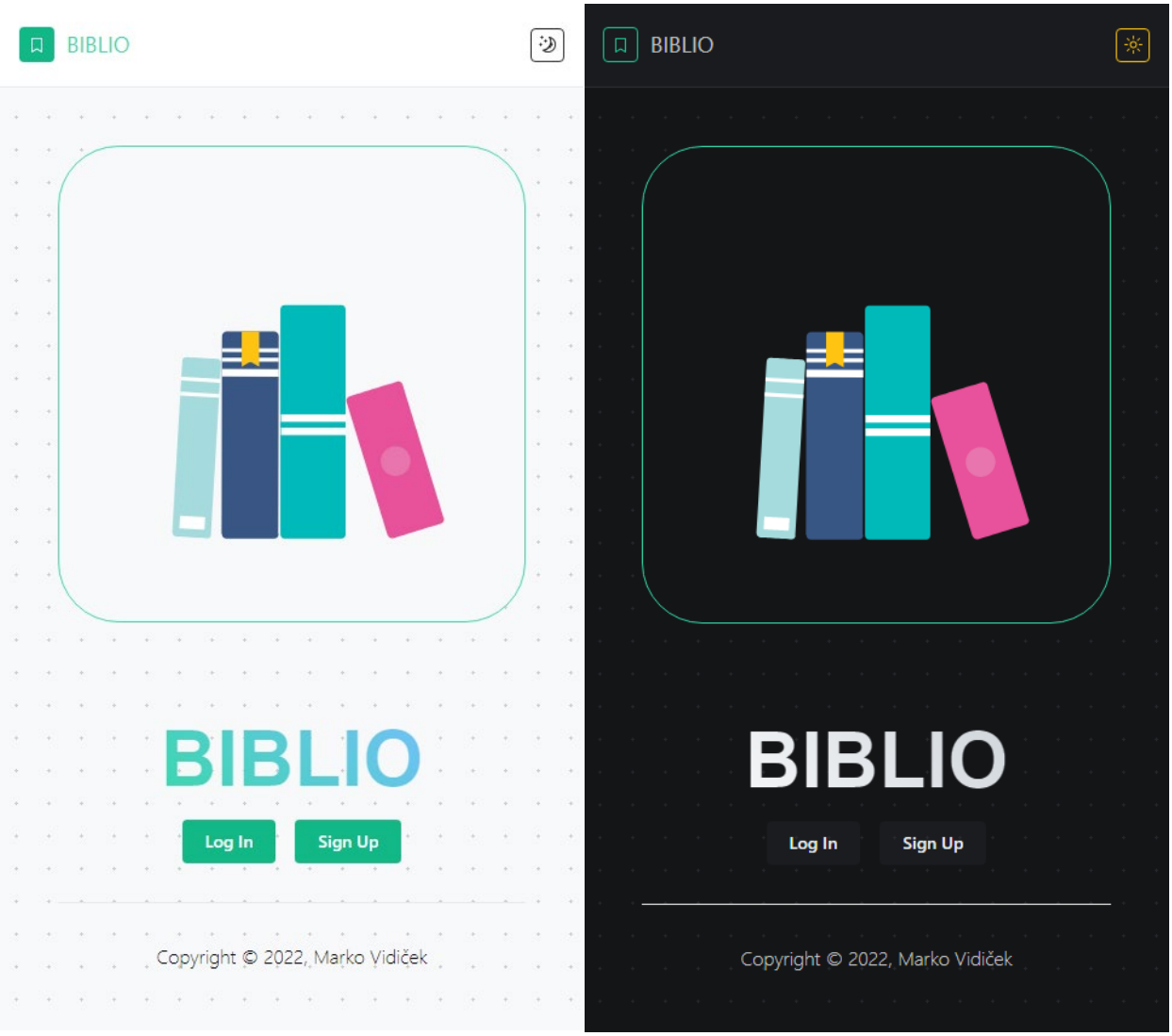
Slika 29. Profile.jsx komponenta

3.9. Dovršena Aplikacija

Ovime su dovršene sve komponente, sljedeće slike prikazuju dovršenu aplikaciju (*light* tema).



Slika 30. Biblio, Dashboard dovršen



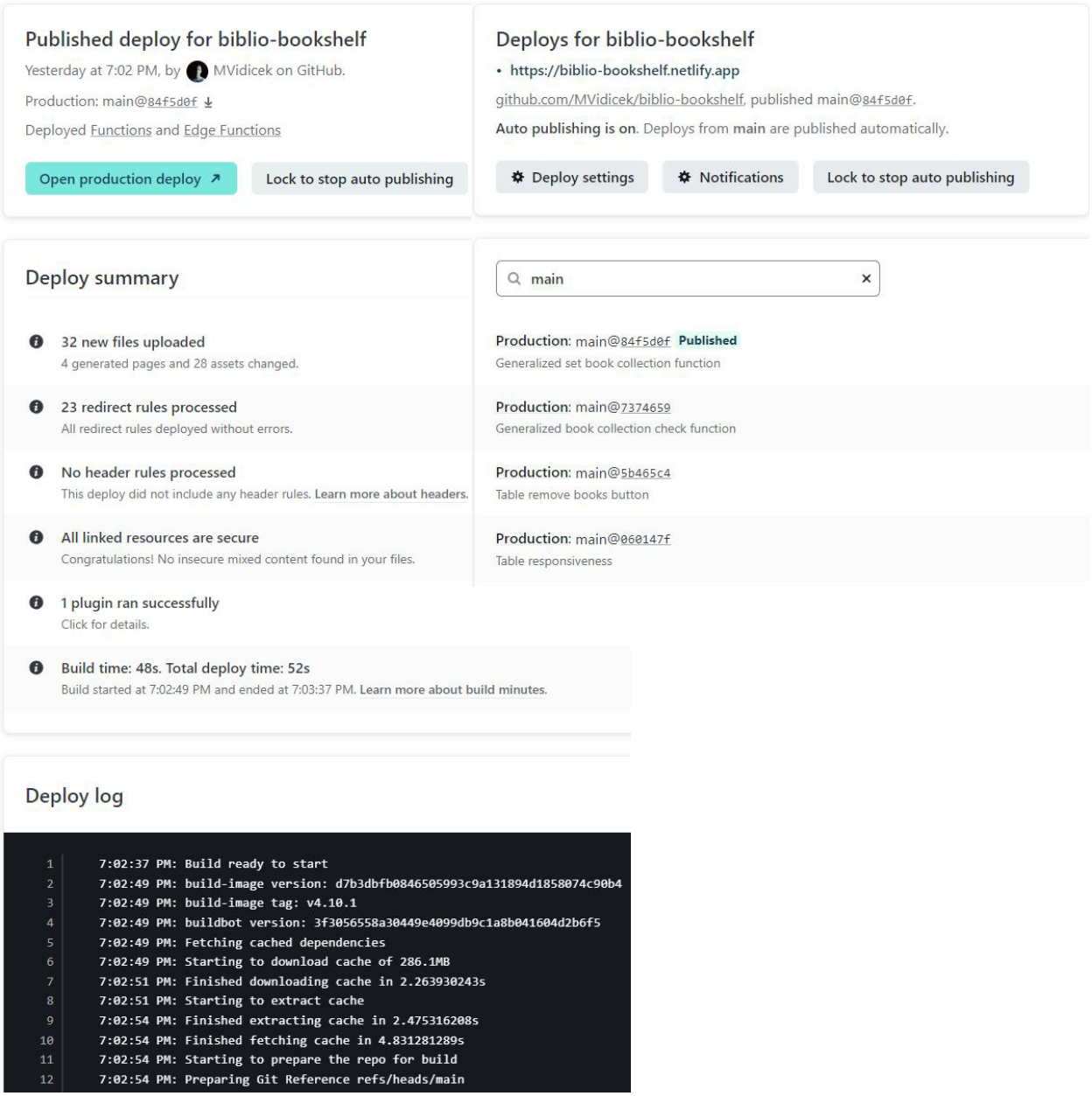
Slika 31. Biblio, Početna dovršena

3.10. Deployment

Sada je samo potrebno postaviti aplikaciju na Netlify poslužitelj koji će izvršiti izgradnju aplikacije preko “build” skripte gdje će Next.js obaviti svoje postupke optimizacije koda svaki puta kada se “pogura” izmjena na GitHub repozitorij (CI/CD, continuous integration & development).

Potrebno je napraviti besplatni Netlify račun i povezati ga s Github računom.

Netlify automatski napravi svoj repozitorij gdje izvršava “build” skriptu i gdje se nalazi aplikacijski kod koji se poslužuje klijentu bilo gdje u svijetu (Slika 32.).



Slika 32. Netlify CD/CI

S ovime su završeni svi koraci u izradi aplikacije, od planiranja, postavljanja backend-a u obliku Google Firebase usluga pa do postavljanja aplikacije na poslužitelj.

Netlify: https://biblio-bookshelf.netlify.app
GitHub: https://github.com/MVidicek/biblio-bookshelf

4. Zaključak

Tehnologije uključene u razvoj web stranica i web aplikacija su se razvile kroz godine do točke gdje individualni programer više ne mora toliko brinuti o funkcionalnostima na niskoj razini već može brzo i efikasno realizirati svoje ideje uz minimalan trošak zbog velikog rasta softvera otvorenog koda koji je besplatan za sve.

Tako je kroz ovaj dan opisan postupak izrade web aplikacije “Biblio” te su opisani svi alati i tehnologije koji su se koristili tijekom izrade.

Naravno, kako bi se pustila veća aplikacija u proizvodnju, potrebna je i veća infrastruktura no za realizaciju testnih aplikacija i osobnih projekata ima i više nego dovoljno besplatnih opcija koje omogućuju sve od planiranja do izgradnje, kontrole verzioniranja i posluživanja aplikacije korisnicima diljem svijeta.

Web razvoj neprestano se razvija s napretkom tehnologije. Iako to možda nećemo moći vidjeti svakodnevno, lako nam je pogledati unatrag i vidjeti kako su se stvari promijenile od početaka do danas

S obzirom na brz razvoj ove industrije i, naravno, konkurenciju, važno je biti u tijeku s najnovijom tehnologijom i alatima kako bismo uvijek bili u najboljem položaju.

Literatura

[1] Povijest web razvoja "https://webdevelopmenthistory.com/timeline/", pristupljeno 15.08.2022
[2] Skriptni jezici na strani poslužitelja "https://htmlcss.fandom.com/wiki/Server-Side_Scripting_Language", pristupljeno 15.08.2022
[3] Web Aplikacije "https://en.wikipedia.org/wiki/Web_application", pristupljeno 15.08.2022
[4] Frontend i Backend "https://en.wikipedia.org/wiki/Frontend_and_backend", pristupljeno 15.08.2022
[5] Javascript "https://en.wikipedia.org/wiki/JavaScript", pristupljeno 17.08.2022
[6] Node.js "https://nodejs.org/en/about/", pristupljeno 17.08.2022
[7] Node Package Manager "https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/", pristupljeno 17.08.2022
[8] Google Firebase "https://www.geeksforgeeks.org/firebase-introduction/", pristupljeno 17.08.2022
[9] React "https://www.geeksforgeeks.org/firebase-introduction/", pristupljeno 17.08.2022
[10] Next.js "https://nextjs.org/learn/foundations/about-nextjs", pristupljeno 17.08.2022
[11] Mantine "https://mantine.dev/pages/about/", pristupljeno 17.08.2022
[12] SWR "https://swr.vercel.app", pristupljeno 17.08.2022
[13] GitHub "https://en.wikipedia.org/wiki/GitHub", pristupljeno 17.08.2022
[14] Netlify "https://en.wikipedia.org/wiki/Netlify", pristupljeno 17.08.2022
[15] Node.js instalacija "https://docs.npmjs.com/downloading-and-installing-node-js-and-npm", pristupljeno 18.08.2022
[16] NPM CLI komande "https://docs.npmjs.com/cli/v6/commands", pristupljeno 18.08.2022
[15] Firebase, postavljanje "https://firebase.google.com/docs/web/setup", pristupljeno 18.08.2022

Slike

Prilozi

Uz ovaj diplomski rad nisu priloženi nikakvi dodatni dokumenti, datoteke ili resursi u digitalnom obliku