GraphQL i React

Full-Stack veb razvoj

Sebastian Grebe

kompjuter biblioteka

Packt›

Izdavač:

kompjuter biblioteka

Obalskih radnika 4a, Beograd

Tel: 011/2520272

e-mail: kombib@gmail.com

internet: www.kombib.rs

Urednik: Mihailo J. Šolajić

Za izdavača, direktor:
Mihailo J. Šolajić

Autor: Sebastian Grebe

Prevod: Slavica Prudkov

Lektura: Miloš Jevtović

Slog: Zvonko Aleksić

Znak Kompjuter biblioteke:
Miloš Milosavljević

Štampa: „Pekograf“, Zemun

Tiraž: 500

Godina izdanja: 2019.

Broj knjige: 518

Izdanje: Prvo

ISBN: 978-86-7310-541-3

Hands-On Full-Stack Web Development with GraphQL and React

Sebastian Grebe

ISBN 978-1-78913-452-0

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд,
се добија на захтев

O AUTORU

Sebastian Grebe je verifikovani stručnjak računarskih nauka za razvoj aplikacija. On je mladi preduzetnik koji razvija različite proizvode. Specijalizovan je za veb razvoj upotrebom modernih tehnologija, kao što su React i FeathersJS, i tradicionalnih tehnologija, kao što su PHP i SQL. Profesionalno se razvio spajanjem starih i novih aplikacija i razvijanjem međuplatformskih aplikacija pomoću React Nativea i Ionica.

Trenutno aktivno radi u svojoj softverskoj agenciji „Open Mind“, koja upravlja različitim softverskim projektima. Takođe aktivno razvija aplikaciju društvene mreže, koja koristi React, Apollo i Cordova, pod nazivom Coupled. Radio je kao softverski inženjer i projekt menadžer za različite kompanije, kao što je DB Netz AG.

Zahvaljujem se mojoj ljubavi što mi je pružila vreme koje mi je bilo potrebno. Njena podrška učinila je svaki dan boljim. Posebno se zahvaljujem mojim roditeljima, koji su me ohrabrivali da krenem sopstvenim putem u životu.

O RECENZENTU

Subhash Shah je šef tehničkog odeljenja u AIMDek Technologies Pvt. Ltdu. On je iskusni arhitekta sa više od 12 godina iskustva, a poseduje i diplomu iz oblasti informacionih tehnologija. Zagovara razvoj otvorenog koda koji koristi za rešavanje važnih poslovnih problema dok smanjuje troškove. Zainteresovan je za mikroservise, analizu podataka, mašinsko učenje, veštačku inteligenciju i baze podataka. Obožava kvalitetni kod i razvoj vođen testiranjem. Njegove tehničke veštine uključuju prevođenje poslovnih zahteva u skalabilnu arhitekturu, dizajniranje održivih rešenja i isporuku projekata. Shah je koautor knjiga „MySQL 8 Administrator’s Guide“ i „Hands-On HighPerformance with Spring 5“, koje je izdala izdavačka kuća „Packt“.

„PACKT“ TRAŽI AUTORE KAO ŠTO STE VI

Ako ste zainteresovani da postanete autor za „Packt“, posetite stranicu authors.packtpub.com i prijavite se što pre. Sarađujemo sa hiljadama programera i tehničkih profesionalaca da bismo im pomogli da podele svoje mišljenje sa globalnom tehničkom zajednicom. Možete da podnesete osnovnu prijavu, da se prijavite za specifičnu temu za koju tražimo autore ili da pošaljete neke svoje ideje.

UVOD

„GraphQL i React za razvoj veb aplikacija“ je knjiga za veb programere koji žele da poboljšaju svoje veštine i izgrade kompletne full-stack aplikacije, koristeći industrijske standarde.

Do kraja ove knjige bićete dovoljno vešti u upotrebi GraphQL-a i Reacta za potrebe razvoja full-stack aplikacije.

U ovoj knjizi ćemo vas voditi kroz implementiranje aplikacija upotrebom Reacta, Apolloa, Node.jsa i SQL-a i opisaćemo najbolju praksu. Takođe ćemo se fokusirati na rešavanje kompleksnih problema pomoću GraphQL-a, kao što su apstrahovanje arhitektura baza podataka koje imaju više tabela i obrada poslatih slika.

ZA KOGA JE OVA KNJIGA

Ovo je knjiga za veb programere koji žele da poboljšaju veštine i izgrade kompletne full-stack aplikacije upotrebom industrijskih standarda. Tipičan čitalac bi bio neko ko želi da istraži kako da koristi GraphQL, React, Node.js i SQL za pisanje celih aplikacija pomoću ovog steka.

ŠTA OBUHVATA OVA KNJIGA

Na početku Poglavlja 1, „Pripremanje razvojnog okruženja“, govorimo o arhitekturi aplikacije opisom osnovnih koncepata i pripremanjem radne postavke Reacta. Videćete kako se uklapaju React i webpack i opisaćemo neke osnovne scenarije upotrebe Reacta. Takođe ćemo pokazati kako da ispravite greške čeone komponente pomoću React Dev Tools alatki.

U Poglavlju 2, „Podešavanje GraphQL-a pomoću Express.jsa“, fokusiraćemo se na Express.js kao primarni sistem za služenje pozadinske komponente. Naučićete kako da upotrebite funkcionalnost rutiranja Express.jsa za implementiranje različitih API-a. Štaviše, na kraju poglavlja ćete podesiti krajnju tačku koja prihvata GraphQL zahteve kroz Apollo Server paket. Da bismo bili sigurni da sve funkcioniše, upotrebićemo Postman za testiranje i potvrđivanje funkcionalnosti pozadinske komponente.

U Poglavlju 3, „Povezivanje sa bazom podataka“, govorimo o tome kako da upotrebite GraphQL za skladištenje podataka i izvršavanje upita za podatke. Kao primer, upotrebljen je tradicionalni SQL za izgradnju cele aplikacije sa MySQL bazom podataka. Da bismo pojednostavili kod baze podataka, koristićemo Sequelize alatku - ona omogućava da izvršavamo upit SQL Servera pomoću regularnog JavaScript objekta i takođe ga čuva otvorenog ako koristimo MySQL, MSSQL, PostgreSQL ili samo SQLite fajl. Izgradićemo modele i šeme za korisnike i postove u Apollou i Sequelizeu.

U Poglavlju 4, „Integrisanje Reacta u pozadinsku komponentu pomoću Apolloa“, objasnićemo kako da povežete Appolo u React i izgradite cele čeone komponente. U ovom poglavlju nisu opisani osnovni tokovi rada Reacta, ali su objašnjene konfiguracije specifične za Apollo.

U Poglavlju 5, „Ponovo upotrebljive React komponente“, detaljno ćemo opisati pisanje složenijih React komponenata i deljenje podataka pomoću njih.

U Poglavlju 6, „Provera identiteta pomoću Apolloa i Reacta“, objasnićemo uobičajene načine provere identiteta korisnika na Vebu i u GraphQL-u i razlike između njih. Vodićemo vas kroz izgradnju kompletnog toka provere identiteta upotrebom najbolje prakse.

U Poglavlju 7, „Obrada slanja slika“, obuhvaćeno je slanje slika pomoću Apolloa i snimanje slika u posebna skladišta objekta, kao što je AWS S3.

U Poglavlju 8, „Rutiranje u Reactu“, opisaćemo kako da implementirate još neke funkcije za krajnjeg korisnika, kao što je stranica profila. To ćemo postići instaliranjem React Routera.

U Poglavlju 9, „Implementiranje renderovanja na strani servera“, objasnićemo zašto je za mnoge aplikacije renderovanje na strani servera obavezno. Važno je za SEO, ali takođe može imati pozitivne efekte na vaše krajnje korisnike. U ovom poglavlju ćemo se fokusirati na prebacivanje aktuelne aplikacije na postavku renderovanja na strani servera.

U Poglavlju 10, „Prijavljivanja u realnom vremenu“, fokusiraćemo na to kako da izgradimo funkcionalnost ćaskanja u realnom vremenu, uključujući i sistem za obaveštenja.

Svake sekunde nova poruka može da pristigne i korisnik može direktno da bude o tome obavešten. Ova funkcionalnost će biti implementirana pomoću manje-više eksperimentalne GraphQL i Apollo funkcije pod nazivom Subscriptions.

U Poglavlju 11, „Pisanje testova“, upotrebićemo Mocha i JavaScript radni okvir za testiranje koda. Fokusiraćemo se primarno na testiranje GraphQL pozadinske komponente i pravilno testiranje React aplikacija.

U Poglavlju 12, „Optimizacija GraphQL-a pomoću Apollo Enginea“, odgovorićemo na pitanja kakva je performansa GraphQL API-a, postoje li neke greške i kako možemo da poboljšamo GraphQL šemu. Za to ćemo upotrebiti Apollo Engine.

U Poglavlju 13, „Kontinualno raspoređivanje pomoću alatki CircleCI i Heroku“, opisaćemo kako se podešava Heroku aplikacija i dobija opcija za izgradnju i raspoređivanje Docker imidža kroz kontinualno raspoređivanje.

DA BISTE DOBILI MAKSIMUM IZ OVE KNJIGE:

Preporučujemo da najpre pročitate prvo poglavlje da biste bili sigurni da razumete osnovne koncepte Reacta i webpacka. Nakon toga, možete da čitate bilo koje poglavlje želite. Sva se samostalna, ali su uređena prema složenosti i mogu zahtevati tehnike opisane u prethodnim poglavljima (aplikacija postaje sve složenija iz poglavlja u poglavlje).

Aplikacija je prilagođena za upotrebu u realnom svetu, ali su neki delovi izostavljeni, kao što su pravilna obrada greške i druge funkcije koje će aplikacije u stvarnom svetu imati, uključujući analitiku. Cilj nam je da vas naučimo različitim tehnikama. Trebalo bi da dobro razumete gradivne blokove i kako se kreiraju veb aplikacije pomoću Reactai GraphQL-a.

Biće vam od velike pomoći ako već duže vremena koristite JavaScript i React, ili, bar, ako imate iskustva sa nekim drugim modernim JavaScript radnim okvirom, jer mnogi koncepti nisu specifični za aplikaciju, već su generalno dobra praksa, kao što je reaktivno renderovanje.

Uglavnom, ovo je knjiga koju možete da upotrebite da biste započeli učenje o React i GraphQL razvoju, fokusirajući se na poglavlja koja vas najviše interesuju.

Preuzimanje fajlova sa primerima koda

Fajlove sa primerima koda za ovu knjigu možete da preuzmete sa našeg sajta:

http://bit.ly/2ZcSl4U

Kada je fajl preuzet, raspakujte ili ekstrahujte direktorijum, koristeći najnoviju verziju:

- WinRAR/7-Zip za Windows
- Zipeg/iZip/UnRarX za Mac
- 7-Zip/PeaZip za Linux

UPOTREBLJENE KONVENCIJE

Postoji veliki broj konvencija teksta koje su upotrebljene u ovoj knjizi.

CodeInText: ukazuje na reči koda u tekstu, nazive tabela baze podataka, nazive direktorijuma, nazive fajlova, ekstenzije fajlova, nazive putanje, skraćene URL-ove, korisnički unos i Twitter postove. Evo i primera: „Prosledićemo prethodno kreirani fajl index.html kao šablon“.

Blok koda je prikazan na sledeći način:

```
state = {
  posts: posts

}
```

Unos u komandnoj liniji napisan je na sledeći način:

```
mkdir ~/graphbook
cd ~/graphbook
```

Zadebljana slova: ukazuju na novi termin, važnu reč ili reči koje vidite na ekranu. Na primer, reči u menijima ili okvirima za dijalog prikazane su u tekstu zadebljanim slovima. Evo i primera: „Nakon toga, kliknite na **Create**“.

Napomene ili važna obaveštenja prikazani su ovako.

Saveti i trikovi su prikazani ovako.

POVRATNE INFORMACIJE

Povratne informacije od naših čitalaca su uvek dobrodošle.

Osnovne povratne informacije: ako imate bilo kakva pitanja o bilo kom aspektu ove knjige, pošaljite nam e-mail na adresu customercare@packtpub.com i napišite naslov knjige.

Štamparske greške: iako smo preduzeli sve mere da bismo obezbedili tačnost sadržaja, moguće su greške. Ako pronađete neku grešku u ovoj knjizi, bili bismo zahvalni ako biste nam to prijavili. Posetite stranicu http://www.packt.com/submit-errata, izaberite knjigu, kliknite na link Errata Submission Form i unesite detalje.

Piraterija: ako pronađete ilegalnu kopiju naših knjiga, u bilo kojoj formi na Internetu, molimo vas da nas o tome obavestite i pošaljete nam adresu lokacije ili naziv veb sajta. Molimo vas, pišite nam na adresu copyright@packt.com i pošaljite nam link ka sumnjivom materijalu.

Ako ste zainteresovani da postanete autor: ako postoji tema za koju ste specijalizovani i zainteresovani ste da pišete ili sarađujete na nekoj od knjiga, pogledajte vodič za autore na adresi authors.packtpub.com.

RECENZIJA

Kada pročitate ovu knjigu, zašto ne biste napisali vaše mišljenje o njoj na sajtu sa kojeg ste je poručili? Potencijalni čitaoci bi mogli da upotrebe vaše mišljenje da bi se odlučili za kupovini knjige, mi u „Packtu“ bismo saznali šta mislite o našim proizvodima, a naši autori mogli bi da vide povratne informacije o svojoj knjizi.

Za više informacija o „Packtu“ posetite sajt packt.com.

1

Priprema razvojnog okruženja

Aplikacija koju ćemo graditi u ovoj knjizi će biti pojednostavljena verzija Facebooka, pod nazivom **Graphbook**.

Kada razvijate aplikaciju, uvek je potrebno da budete dobro pripremljeni. Međutim, pre nego što počnemo rad, treba da spojimo stek. U ovom poglavlju ćemo istražiti da li naša tehnika funkcioniše dobro u procesu razvoja, šta nam je potrebno da bismo započeli rad i koje alatke nam mogu pomoći kada gradimo softver.

Objasnićemo arhitekturu za aplikaciju opisom osnovnih koncepata, kompletnog procesa i pripreme za radnu postavku Reacta.

U ovom poglavlju obrađene su sledeće teme:

- arhitektura i tehnologija
- kritičko razmišljanje o načinu projektovanja steka
- izgradnja React i GraphQL steka
- instaliranje i konfigurisanje Node.jsa
- podešavanje React razvojnog okruženja pomoću webpacka i Babela i drugi zahtevi
- ispravljanje grešaka React aplikacija pomoću alatki Chrome DevTools i React DeveloperTools
- upotreba alatke `webpack-bundle-analyzer` za proveru veličine paketa

ARHITEKTURA APLIKACIJE

Od prvog izdanja (2015. godine) GraphQL je postao nova alternativa za standardne SOAP i REST API-e. GraphQL je specifikacija, kao što su SOAP i REST, koju možete da pratite da biste strukturirali aplikaciju i tok podataka. GraphQL je inovativan, jer omogućava da izvršavate upite za specifična polja entiteta, kao što su korisnici i postovi. Ova funkcionalnost čini ga veoma dobrim za usmeravanje ka više platformi istovremeno. Aplikacije za mobilne uređaje možda ne zahtevaju da u njima budu prikazani svi podaci koji su prikazani unutar pretraživača na desktop računaru. Upit koji šaljete se sastoji od objekta sličnog JSON-u, koji definiše koje informacije platforma zahteva. Na primer, upit za post može izgledati ovako:

```
post {
  id
  text
  user {
    user_id
    name
  }
}
```

GraphQL razrešava korektne entitete i podatke, kao što je specifikovano u objektu upita. Svako polje u GraphQL-u predstavlja funkciju koja se razrešava na vrednost. Te funkcije se nazivaju Resolver funkcije. Vraćena vrednost može da bude samo vrednost odgovarajuće baze podataka, kao što je naziv korisnika, ili može da bude datum, koji je formatirao server pre nego što ga je vratio.

GraphQL je potpuno nezavisan od baze podataka i može da se implementira u bilo koji programski jezik.

Mi ćemo izostaviti korak implementacije sopstvene GraphQL biblioteke, pa ćemo upotrebiti Apollo, koji je GraphQL server za Node.js ekosistem. Zahvaljujući Apollo timu, veoma je modularan. Apollo funkcioniše sa mnogim uobičajenim Node.js radnim okvirima, kao što su Hapi, Koa i Express.js.

Mi ćemo upotrebiti Express.js kao osnovu, jer je popularan u Node.js i GraphQL zajednicama.

GraphQL može da se upotrebi sa više sistema baze podataka i sa distribuiranim sistemima za obezbeđivanje jasnog API-a za sve servise. Omogućava programerima da ujednače postojeće sisteme i da obrađuju podatke preuzete za klijent aplikacije.

Od vas zavisi kako ćete kombinovati baze podataka, eksterne sisteme i druge servise u pozadinskoj komponenti jednog servera.

U ovoj knjizi mi ćemo upotrebiti MySQL server kao skladište podataka pomoću alatke Sequelize.

SQL je najpoznatiji i najčešće upotrebljavan jezik za upite u bazu podataka, a, zahvaljujući Sequelizeu, imamo modernu klijent biblioteku za Node.js server za povezivanje sa SQL serverom.

HTTP je standardni protokol za pristup GraphQL API-u. Takođe se primenjuje na Apollo Servere. Međutim, GraphQL nije fiksiran za jedan mrežni protokol.

Izgradićemo čeonu komponentu Graphbook aplikacije pomoću Reacta. React je JavaScriptUI radni okvir (izdao ga je „Facebook“) - on predstavlja mnoge tehnike koje se ne koriste uobičajeno za izgradnju interfejsa na Vebu kao i u izvornim okruženjima.

Upotreba Reacta ima mnogo značajnih prednosti. Kada gradite React aplikaciju, uvek ćete rastaviti kod na više komponenata, poboljšavajući tako efikasnost i povećavajući mogućnost ponovne upotrebe. Naravno, možete to da uradite bez upotrebe Reacta, ali on olakšava ovaj proces. Štaviše, uči vas kako da reaktivno ažurirate stanja aplikacije kao i UI. Nikada nemojte da ažurirate UI, a zatim podatke posebno.

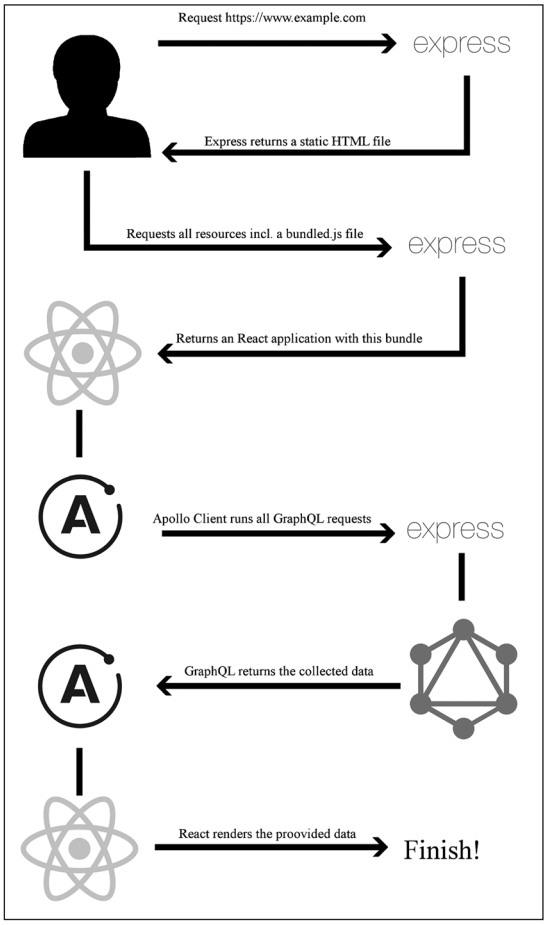
React čini renderovanje veoma efikasnim upotrebom virtuelnog DOM-a, koji upoređuje virtuelni i aktuelni DOM i ažurira ga u skladu sa tim. Samo kada postoji razlika između virtuelnog i realnog DOM-a React će primeniti nove promene. Ova logika zaustavlja pretraživač od ponovnog izračunavanja rasporeda, Cascading Style Sheetove i druga izračunavanja koja negativno utiču na uopštenu performansu aplikacije.

U ovoj knjizi ćemo upotrebiti Apollo klijent biblioteku. Ona se, naravno, integriše sa Reactom i Apollo serverom.

Ako sve ovo spojite, rezultat će biti glavni stek koji se sastoji od elemenata Node.js, Express.js, Apollo, SQL, Sequelize i React.

Osnovna postavka

Osnovna postavka koja doprinosi da aplikacija funkcioniše je logički tok zahteva, koji izgleda kao na sledećoj slici.



Evo kako funkcioniše logički tok zahteva:

1. Klijent šalje zahtev za sajt.
2. Express.js server obrađuje ovaj zahtev i služi statički HTML fajl.
3. Klijent preuzima sve potrebne fajlove, u skladu sa ovim HTML fajlom. Fajlovi uključuju i prateći JavaScript fajl.

4. Ovaj prateći JavaScript fajl je naša React aplikacija. Nakon izvršenja celog JavaScript koda iz ovog fajla, svi potrebni Ajax alias GraphQL zahtevi izvršeni su na Apollo serveru.
5. Express.js prima zahteve i prosleđuje ih u Apollo krajnju tačku.
6. Apollo izvršava upit za sve potrebne podatke iz svih dostupnih sistema, kao što su SQL server ili nezavisni servisi, spaja podatke i šalje ih nazad kao JSON.
7. React može da renderuje JSON podatke u HTML.

Ovaj tok rada je osnovna postavka koja čini da aplikacija funkcioniše. U nekim slučajevima ima smisla klijentu ponuditi renderovanje na strani servera. Server treba da renderuje i sam pošalje sve `XMLHttpRequests` zahteve pre vraćanja HTML-a klijentu. Korisnik će uštedeti nekoliko koraka ako server šalje zahteve za početno učitavanje. Fokusiraćemo se na ovu temu u sledećem poglavlju, ali, ukratko da kažem, to je arhitektura aplikacije. Imajući to na umu, sada ćemo podesiti razvojno okruženje.

INSTALIRANJE I KONFIGURISANJE NODE.JS-a

Prvi korak pripreme za projekat je da instaliramo Node.js. Postoji dva načina da to uradimo:

- -Jedna opcija je da instaliramo **Node Version Manager (NVM)**. Prednost upotrebe NVM-a je što možemo da pokrenemo više verzija Node.jsa jednu pored druge i to će izvršiti proces instalacije na skoro svakom sistemu zasnovanom na UNIX-u, kao što su Linux i macOS. U ovoj knjizi nije nam potrebna opcija za prekopčavanje između različitih verzija Node.jsa.
- -Druga opcija je da instaliramo Node.js pomoću upravljača paketima za distribuciju ako koristimo Linux. Zvanični PKG fajl je za Mac, dok je MSI fajl za Windows. Mi ćemo upotrebiti regularni Linux upravljač paketima za ovu knjigu, jer je to najjednostavniji metod.

Možete pronaći **Downloads** odeljak Node.jsa na adresi https://nodejs.org/en/download/.

Upotrebićemo prethodno opisanu drugu opciju. Ona je jasna i obuhvata regularne konfiguracije servera. Ja ću ovaj proces opisati što je moguće kraće i izostaviću sve druge opcije, kao što su Chocolatey za Windows ili Brew za Mac, koje su specijalizovane za te specifične operativne sisteme.

Pretpostavljam da koristite sistem zasnovan na Debianu zbog lakoće upotrebe, jer ima normalan APT upravljač paketima i skladišta za jednostavno instaliranje Node.jsa i MySQL-a. Ako ne koristite sistem zasnovan na Debianu, možete da potražite odgovarajuće komande za instaliranje Node.jsa na adresi https://nodejs.org/en/download/package-manager/.

Naš projekat će biti nov, tako da možemo da upotrebimo Node.js 10 bez ikakvih problema. Možete da izostavite sledeću instalaciju Node.jsa ako imate verziju 6 ili noviju:

1. Prvo ćemo dodati odgovarajuće skladište za upravljač paketima, tako što ćemo pokrenuti sledeću komandu:

   ```
   curl –sL https://deb.nodesource.com/setup_10.x | sudo –E bash –
   ```

2. Zatim ćemo instalirati Node.js i alatke za izgradnju za izvorne module pomoću sledeće komande:

   ```
   sudo apt-get install –y nodejs build-essential
   ```

3. Na kraju ćemo otvoriti terminal i verifikovati da je instalacija uspešna:

   ```
   node --version
   ```

Instalacija Node.jsa pomoću upravljača paketima će automatski instalirati npm.

Odlično! Sada ste spremni da pokrenete JavaScript na strani servera pomoću Node.jsa i da instalirate Node.js module za projekte, koristeći npm.

Sve zavisnosti na koje se projekat oslanja su dostupne na adresi https://npmjs.com i mogu da budu instalirane pomoću npm-a ili Yarna, ako vam ove alatke više odgovaraju.

PODEŠAVANJE REACTA

Razvojno okruženje za projekat je spremno. U ovom odeljku ćemo instalirati i konfigurisati React, što je i primarni zadatak u ovoj knjizi. Prvo ćemo kreirati novi direktorijum za naš projekat:

```
mkdir ~/graphbook
cd ~/graphbook
```

Naš projekat će koristiti Node.js i mnoge npm pakete. Kreirajte `package.json` fajl da biste instalirali i upravljali svim zavisnostima za projekat.

Ovaj fajl skladišti informacije o projektu, kao što su broj verzije, naziv, zavisnosti i tako dalje.

Samo pokrenite komandu `npm init` da biste kreirali prazan `package.json` fajl:

```
npm init
```

Npm će postaviti neka pitanja, kao, na primer, koji je naziv paketa, što je, u stvari, naziv projekta. Unesite `Graphbook` da biste uneli naziv aplikacije u generisani `package.json` fajl.

Ja najradije počinjem sa brojem verzije 0.0.1, jer podrazumevani broj verzije koji npm nudi je 1.0.0, što za mene predstavlja prvu stabilnu verziju. Međutim, vi ćete se sami opredeliti koju verziju ćete upotrebiti.

Možete preskočiti sva druga pitanja pomoću tastera *Enter* da biste sačuvali podrazumevane vrednosti npm-a. Većina ovih vrednosti nije važna, jer obezbeđuje informacije kao što su opis ili link ka skladištu. Mi ćemo popuniti ostala polja, kao što su skriptovi dok obrađujemo primer iz ove knjige. Možete da vidite primer komandne linije na sledećoj slici.

```
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (graphbook)
version: (1.0.0) 0.0.1
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\Users\sebig\Desktop\testit\graphbook\package.json:

{
  "name": "graphbook",
  "version": "0.0.1",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Prva i najvažnija zavisnost za ovu knjigu je React. Upotrebite npm za dodavanje Reacta u projekat:

```
npm install --save react react-dom
```

Ova komanda instalira dva npm paketa sa adrese https://npmjs.com u direktorijum projekta pod `node_modules`.

Npm automatski edituje `package.json` fajl, jer smo upotrebili opciju --save i dodali te pakete sa poslednjim dostupnim brojevima verzije.

Možda se pitate zašto smo instalirali dva paketa, mada nam je potreban samo React. Paket react obezbeđuje samo metode specifične za React. React veze, kao što su `componentDidMount`, `componentWillReceivesProps` i Reactova component klasa, dolaze iz ovog paketa, koji vam je potreban da biste mogli da pišete React aplikacije.

U većini slučajeva nećete ni primetiti da ste upotrebili `react-dom`. Ovaj paket sadrži sve funkcije za povezivanje aktuelnog DOM-a pretraživača sa React aplikacijom. Obično ćete samo jednom u kodu upotrebiti ReactDOM.render za renderovanje aplikacije na specifičnu tačku u HTML-u. Mi ćemo opisati renderovanje Reacta u sledećem poglavlju.

Postoji i funkcija pod nazivom `ReactDOM.findDOMNode`, koja vam daje direktan pristup za `DOMNode`, ali ne savetujem da je upotrebite, jer promene u `DOMNode` funkciji nisu dostupne u samom Reactu. Nikada nisam imao potrebu da koristim ovu funkciju, pa pokušajte i vi da je izbegavate, ukoliko je to moguće.

Pripremanje i konfigurisanje webpacka

Naš pretraživač zahteva `index.html` fajl kada pristupa aplikaciji. Pretraživač specifikuje sve fajlove koji su potrebni za pokretanje aplikacije. Treba da kreiramo `index.html` fajl, koji ćemo služiti kao tačku unosa aplikacije:

1. Kreirajte poseban direktorijum za fajl index.html:

```
mkdir public
touch index.html
```

2. Zatim, snimite ovo unutar fajla index.html:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-
      scale=1.0">
    <meta http-equiv="X-UA-Compatible"
    content="ie=edge">
    <title>Graphbook</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

Kao što možete da vidite, ovde nije učitan JavaScript. Postoji samo `div` sa `root` id-jem. Ova div oznaka je `DOMNode` – u nju će našu aplikaciju renderovati `ReactDOM`.

Dakle, kako ćemo dobiti React pokrenut pomoću ovog `index.html` fajla?

Da bismo to postigli, potrebno je da upotrebimo paker modula veb aplikacije. On priprema i pakuje sve što pripada aplikaciji. Svi potrebni JavaScript fajlovi i `node_modules` su upakovani i minimizirani; SASS i SCSS pretprocesori su prevedeni u CSS i spojeni su i minimizirani.

Neki od pakera modula su webpack, Parcel i Gulp. Za naš primer ćemo upotrebiti webpack. To je najčešće upotrebljavan paker modula, koji ima veliku zajednicu koja ga okružuje. Da bismo spakovali JavaScript kod, treba da instaliramo webpack i sve njegove zavisnosti na sledeći način:

```
npm install --save-dev @babel/core babel-eslint babel-loader @
babel/preset- env @babel/preset-react clean-webpack-plugin css
-loader eslint file-loader html-webpack-plugin style-loader url
-loader webpack webpack-cli webpack- dev-server @babel/plugin
-proposal-decorators @babel/plugin-proposal- function-sent @
babel/plugin-proposal-export-namespace-from @babel/plugin
- proposal-numeric-separator @babel/plugin-proposal
-throw-expressions

@babel/plugin-proposal-class-properties
```

Ova alatka dodaje sve razvojne alatke u `devDependencies`, koji se nalazi u package.json fajlu koji treba da omogući pakovanje aplikacije. Alatke su instalirane samo u razvojnom okruženju, a izostavljene su u proizvodnji.

Kao što možete da vidite iz prethodnog koda, takođe smo instalirali eslint, koji prolazi kroz kod u toku rada i proverava greške. Potreban nam je eslint konfiguracioni fajl, koji ćemo instalirati sa adrese https://npmjs.com. Sledeća korisna prečica instalira `eslint` konfiguraciju koju su kreirali ljudi u „Airbnbu“, uključujući i sve zavisnosti ravnopravnosti. Izvršite ih odmah:

```
npx install-peerdeps --dev eslint-config-airbnb
```

Kreirajte .eslintrc fajl u rootu direktorijuma projekta za upotrebu airbnb konfiguracije:

```
{
  "extends": ["airbnb"],
  "env": {
    "browser": true,
    "node": true
  },
  "rules": {
    "react/jsx-filename-extension": "off"
  }
}
```

Ukratko, ovaj .eslinrc fajl učitava airbnb konfiguraciju; mi definišemo okruženja gde će se naš kod pokrenuti i isključujemo jedno podrazumevano pravilo.

Pravilo react/jsx-filename-extension izbacuje upozorenje kada koristimo JSX sintaksu unutar fajla, koja se ne završava sa .jsx. Naši fajlovi će se završavati sa .js, pa ćemo uključiti ovo pravilo.

Podešavanje webpacka je malo komplikovano. Postoje mnoge opcije koje mogu da ometaju jedna drugu i dovedu do problema kada pakujete aplikaciju. Kreiraćemo sada webpack.client.config.js fajl u root direktorijumu projekta.

Unesite sledeće:

```
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const CleanWebpackPlugin = require('clean-webpack-plugin');
const buildDirectory = 'dist';
const outputDirectory = buildDirectory + '/client';
module.exports = {
  mode: 'development',
  entry: './src/client/index.js',
  output: {
    path: path.join( dirname, outputDirectory),
    filename: 'bundle.js'
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        use: {
          loader: 'babel-loader'
        }
      },
```

```
        {
          test: /\.css$/,
          use: ['style-loader', 'css-loader']
        }
      ]
    },
    devServer: {
      port: 3000,
      open: true
    },
    plugins: [
        new CleanWebpackPlugin({
          cleanOnceBeforeBuildPatterns: [path.join( dirname,
          buildDirectory)]
        }),
        new HtmlWebpackPlugin({
          template: './public/index.html'
        })
    ]
};
```

Webpack konfiguracioni fajl je samo regularni JavaScript fajl, u kojem možete da zahtevate `node_modules` i prilagođene JavaScript fajlove. To je potpuno isto kao i bilo gde unutar Node.jsa. Sada ćemo ukratko opisati sva glavna svojstva ove konfiguracije. Razumevanje ovih svojstava će doprineti da prilagođene webpack konfiguracije budu mnogo jednostavnije. Sve važne tačke su opisane:

- `HtmlWebpackPlug` - Automatski generiše HTML fajl koji uključuje sve webpack komplete. Prosledićemo prethodno kreirani index.html kao šablon.
- `CleanWebpackPlugin` - Prazni sve obezbeđene direktorijume da bi izbrisao stare gradivne fajlove. Svojstvo `cleanOnceBeforeBuildPatterns` specifikuje niz direktorijuma koji su izbrisani pre početka procesa izgradnje.
- Polje `entry` ukazuje webpacku gde se nalazi početna tačka aplikacije. Ovaj fajl treba da kreiramo.
- Objekat `output` specifikuje kako se naziva paket i gde bi trebalo da bude snimljen. Za ovaj primer to je `dist/client/bundle.js`.
- Unutar `module.rules` svojstva mi usklađujemo ekstenzije fajla sa odgovarajućim programima za učitavanje. Sve JavaScript fajlove (osim onih koji se nalaze u modulu `node_modules`) prevodi Babel, specifikovano u babel-loaderu, da bismo mogli da upotrebimo ES6 funkcije unutar koda. CSS će obraditi `style-loader` i `css-loader`. Postoje mnogi programi za učitavanje za JavaScript, CSS i druge dostupne ekstenzije fajlova.

- Funkcija `devServer` webpacka omogućava da se pokrene direktno React kod. Uključuje aktivni kod za ponovno učitavanje u pretraživač bez ponovnog pokretanja verzije ili „osvežavanja“ kartice pretraživača.

Ako vam je potreban detaljniji pregled webpack konfiguracije, pogledajte zvaničnu dokumentaciju na adresi https://github.com/webpack/docs/wiki/configuration.

Nedostaje nam `src/client/index.js` fajl iz webpack konfiguracije, pa ćemo ga sada kreirati:

```
mkdir src/client
cd src/client
touch index.js
```

Možete da ostavite fajl prazan za trenutak. Ovaj fajl može da spakuje webpack bez sadržaja unutar njega. Mi ćemo ga popuniti u ovom poglavlju.

Da bismo pokrenuli razvojni webpack server, dodaćemo komandu u fajl `package.json`, koji možemo da pokrenemo koristeći `npm`.

Dodajte sledeću liniju u objekat scripts unutar fajla `package.json`:

```
"client": "webpack-dev-server --devtool inline-source-map --hot
--config webpack.client.config.js"
```

Sada izvršite komandu `npm run client` u konzoli i vidite kako se otvara novi prozor pretraživača. Mi pokrećemo `webpack-dev-server` sa novokreiranim konfiguracionim fajlom.

Naravno, pretraživač je i dalje prazan, ali ako pogledate HTML pomoću Chrome DevTools alatki, možete da vidite da već imamo `bundle.js` fajl, a da je `index.html` fajl upotrebljen kao šablon.

Uključili smo prazan `index.js` fajl u paket i možemo da ga pošaljemo u pretraživač. Zatim ćemo renderovati prvu React komponentu unutar šablonskog index.js fajla.

Renderovanje prve React komponente

Postoje mnoge dobre tehnike za React. Osnovna namena upotrebe Reacta je da rastavimo kod u posebne komponente gde god je to moguće. Opisaćemo ovaj pristup detaljnije u Poglavlju 5, „Ponovno upotrebljive React komponente“.

Naš `index.js` fajl je glavna početna tačka koda čeonog prikaza i tako bi trebalo i da ostane. Nemojte uključivati poslovnu logiku u ovaj fajl. Umesto toga, učinite da ostane čist i jednostavan.

Fajl `index.js` treba da uključuje sledeći kod:

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(<App/>, document.getElementById('root'));
```

Izdanje *ECMAScript 2015* predstavilo je funkciju `import`. Upotrebićemo je da izvršimo zahtev za `npm` pakete, `react` i `react-dom`, i za našu prvu React komponentu, koju ćemo sada napisati.

Naravno, važno je da obuhvatimo primer programa `Hello World`.

Kreirajte `App.js` fajl pored fajla index.js, sa sledećim sadržajem:

```
import React, { Component } from 'react';

export default class App extends Component {
  render() {
    return (
      <div>Hello World!</div>
    )
  }
}
```

Ova klasa je eksportovana, a zatim ju je fajl `index.js` importovao.

Kao što je ranije opisano, sada aktivno koristimo `ReactDOM`.render u fajlu `index.js`.

Prvi parametar `ReactDOM`.rendera je komponenta koju želimo da renderujemo, a to je klasa `App` koja prikazuje poruku **Hello World!**. Drugi parametar je `DOMNode` pretraživača, gde bi komponenta trebalo da se renderuje. Mi ćemo primiti `DOMNode` sa čistim document.`getElementById` JavaScriptom.

Definisali smo osnovni element kada smo ranije kreirali `index.html` fajl. Nakon snimanja `App.js` fajla, webpack će ponovo pokušati da izgradi sve. Međutim, ne bi trebalo da može to da uradi. Webpack će naići na problem pakovanja index.js fajla, zbog <App/> tag sintakse koju koristimo u metodu `ReactDOM.render`, koji nije preveden u normalnu JavaScript funkciju.

Konfigurisali smo webpack za učitavanje Babela za JS fajl, ali nismo rekli Babelu šta da prevede, a šta da ne prevede.

Kreiraćemo .babelrc fajl u root direktorijumu sa sledećim sadržajem:

```
{
  "plugins": [
    ["@babel/plugin-proposal-decorators", { "legacy": true }],
    "@babel/plugin-proposal-function-sent",
    "@babel/plugin-proposal-export-namespace-from",
    "@babel/plugin-proposal-numeric-separator",
    "@babel/plugin-proposal-throw-expressions",
    ["@babel/plugin-proposal-class-properties", { "loose": false }]
  ],
  "presets": ["@babel/env","@babel/react"]
}
```

Možda će biti potrebno da restartujete server, jer .babelrc fajl nije ponovo učitan kada se desila promena u fajlu. Nakon nekoliko trenutaka, videćete standardnu **Hello World!** poruku u pretraživaču.

Ovde smo ukazali Babelu da treba da upotrebi `@babel/preset-env` i `@babel/preset-react`, instalirane zajedno sa webpackom. Ove unapred podešene vrednosti omogućavaju Babelu da transformiše specifičnu sintaksu, kao što je JSX, koju koristimo za kreiranje normalnog JavaScripta koji razumeju svi pretraživači i koji webpack može da spakuje. Štaviše, mi koristimo neke Babel dodatne module koje smo instalirali, jer oni transformišu specifičnu sintaksu koju unapred podešene vrednosti ne obuhvataju.

Renderovanje nizova iz React stanja

`Hello World!` je primer koji se mora primeniti u svakoj dobroj knjizi o programiranju, ali to nam nije cilj kada koristimo React.

Društvena mreža, kao što su Facebook ili Graphbook, koju trenutno pišemo, zahteva izvor vesti i unos za postavljanje vesti. Hajde sada da to implementiramo.

Zbog jednostavnosti prvog poglavlja, to ćemo uraditi unutar `App.js` fajla.

Treba da upotrebimo neke lažne podatke, jer još nismo podesili GraphQL API. Kasnije ih možemo zameniti pravim podacima.

Definišite novu promenljivu iznad klase `App` na sledeći način:

```
const posts = [{
  id: 2,
  text: 'Lorem ipsum',
  user: {
    avatar: '/uploads/avatar1.png',
    username: 'Test User'
  }
},
{
  id: 1,
  text: 'Lorem ipsum',
  user: {
    avatar: '/uploads/avatar2.png',
    username: 'Test User 2'
  }
}];
```

Sada ćemo renderovati ova dva lažna posta u Reactu.

Zamenite aktuelni sadržaj metoda `render` sledećim kodom:

```
const { posts } = this.state;

return (
  <div className="container">
    <div className="feed">
      {posts.map((post, i) =>
        <div key={post.id} className="post">
          <div className="header">
            <img src={post.user.avatar} />
            <h2>{post.user.username}</h2>
          </div>
          <p className="content">
            {post.text}
          </p>
        </div>
      )}
    </div>
  </div>
)
```

Mi ponavljamo niz `posts` pomoću funkcije `map`, koja ponovo izvršava internu funkciju povratnog poziva, prosleđujući svaku stavku niza kao parametar, jednu po jednu. Drugi parametar se naziva i, a predstavlja indeks elementa niza koji obrađujemo. Zatim, React renderuje sve što se vraća iz funkcije `map`.

Mi samo vraćamo HTML postavljanjem podataka svakog posta u ES6 velike zagrade. Velike zagrade ukazuju Reactu da treba da interpretira i proceni kod unutar njih kao JavaScript.

Kao što možete da vidite u prethodnom kodu, mi ekstrahujemo postove koje želimo da renderujemo iz stanja komponente destrukturiranjem. Ovaj tok podataka je veoma pogodan, jer možemo da ažuriramo stanje bilo kada kasnije u aplikaciji, a postovi će biti renderovani.

Da bismo dobili postove, možemo da ih definišemo unutar klase pomoću **inicijalizatora svojstva**. Dodajte sledeći kod na početak klase `App`:

```
state = {
  posts: posts
}
```

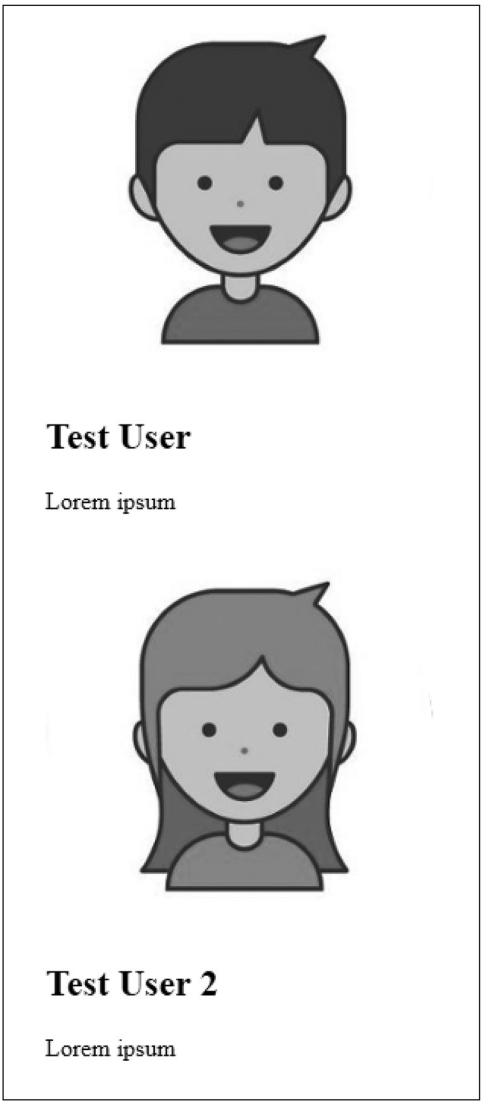
Stariji način implementiranja ovog zadatka, bez upotrebe ES6 funkcije, bio je kreiranje konstruktora:

```
constructor(props) {
  super(props);


  this.state = {
    posts: posts
  };
}
```

Prilikom inicijalizacije klase `App` postovi će biti ubačeni u stanje klase i renderovani. Važno je da se prijavite kao korisnik `super` da biste mogli da pristupite ovom zadatku.

Prethodni metod je mnogo jasniji i preporučujem ga zbog čitljivosti. Kada snimate kod, trebalo bi da vidite renderovane postove. Oni treba da izgledaju kao na sledećoj slici.



Izvor: https://www.vecteezy.com/

Slike koja ja ovde koristim su dostupne besplatno. Možete da upotrebite bilo koji drugi materijal koji imate, sve dok se putanja poklapa sa znakovnim nizom iz niza `posts`. Možete da pronađete te slike u zvaničnom GitHub skladištu za ovu knjigu.

CSS sa webpackom

Postovi sa prethodne slike još nisu dizajnirani. Već sam dodao CSS klase u HTML koji naša komponenta vraća.

Za bolji izgled postova možemo da upotrebimo CSS u JS-u, koristeći pakete kao što je styled-component, koji je React paket. Na primer, druge alternative uključuju Glamorous i Radium. Postoje brojni razlozi zašto ne prelazimo na takav tok rada i zašto se držimo „dobrog starog" CSS-a. Pomoću drugih alatki ne možemo efikasno da upotrebimo SASS, SCSS ili LESS. Ja treba da radim zajedno sa drugim ljudima, kao što su grafički dizajneri, koji mogu da obezbede i upotrebe CSS, ali ne i program styled-components. Uvek postoji prototip ili postojeći CSS koji može da se upotrebi, pa zašto bih trošio vreme prevođenjem toga u styled-components CSS kada mogu da nastavim da koristim standardni CSS?

Ovde ne postoji dobra ili loša opcija; slobodno možete da implementirate stil na način na koji želite. Međutim, u ovoj knjizi mi ćemo se držati „dobrog starog" CSS-a.

U našem `webpack.client.config.js` fajlu smo već specifikovali CSS pravilo, kao što je prikazano u sledećem isečku koda:

```
{
  test: /\.css$/,
  use: ['style-loader', 'css-loader'],
},
```

Komanda `style-loader` injektira upakovani CSS direktno u DOM. A komanda `css-loader` će razrešiti sve primere komandi `import` ili `url` u CSS kodu.

Kreirajte fajl `style.css` u direktorijumu `/assets/css` i popunite sledeće:

```
body {
  background-color: #f6f7f9;
  margin: 0;
  font-family: 'Courier New', Courier, monospace
}
p {
  margin-bottom: 0;
}
.container {
  max-width: 500px;
  margin: 70px auto 0 auto;
}
.feed {
  background-color: #bbb;
  padding: 3px;
```

```
  margin-top: 20px;
}
.post {
  background-color: #fff;
  margin: 5px;
}
.post .header {
  height: 60px;
}
.post .header > * {
  display: inline-block;
  vertical-align: middle;
}
.post .header img {
  width: 50px;
  height: 50px;
  margin: 5px;
}
.post .header h2 {
  color: #333;
  font-size: 24px;
  margin: 0 0 0 5px;
}
.post p.content {
  margin: 5px;
  padding: 5px;
  min-height: 50px;
}
```

Ako „osvežimo“ prikaz u pretraživaču, ostaće isti stari HTML kao i ranije.

Ovaj problem se javlja zato što webpack paker modula i ne „zna“ ništa o CSS-u; poznaje samo JavaScript. Treba da importujemo CSS fajl negde u kod.

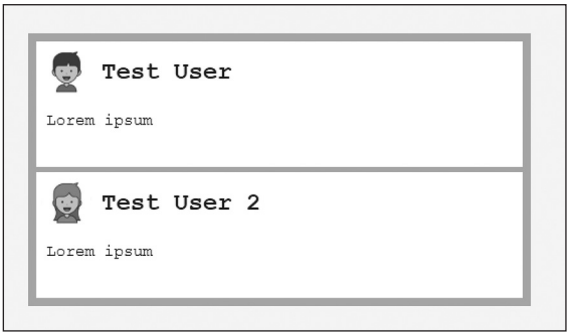
Umesto da upotrebimo `index.html` i dodamo oznaku head, možemo da upotrebimo webpack i CSS pravilo za učitavanje direktno u fajl `App.js`. Ovo rešenje je veoma pogodno, jer će sav potreban CSS kroz aplikaciju postati minimiziran i upakovan. Webpack automatizuje ovaj proces.

U `App.js` fajl dodajte sledeći kod iza React `import` iskaza:

```
import '../../assets/css/style.css';
```

Webpack magično ponovo izgrađuje paket i „osvežava“ karticu pretraživača.

Uspešno smo renderovali lažne podatke pomoću Reacta i stilizovali ih pomoću pratećeg CSS-a iz webpacka. Sada bi trebalo da vidite nešto slično kao na sledećoj slici.



Izvor: https://www.vecteezy.com/

Rezultat već izgleda dobro.

Obrada događaja i ažuriranje stanja pomoću Reacta

Na početku ovog projekta bilo bi dobro da imate jednostavnu oblast `textarea` gde možete da kliknete na dugme, a zatim da novi post bude dodat u statički niz `posts` koji je napisan u klasi `App`.

Dodajte sledeći kod iznad oznake div u klasi `feed`:

```
<div className="postForm">
 <form onSubmit={this.handleSubmit}>
  <textarea value={postContent} onChange={this.handlePostContentChange}
    placeholder="Write your custom post!"/>
       <input type="submit" value="Submit" />
   </form>
  </div>
```

Možemo da upotrebimo obrasce u Reactu bez ikakvih problema. React može da presretne poslati događaj zahteva, obezbeđujući obrascu svojstvo `onSubmit`, koje će biti funkcija za obradu logike obrasca.

Prosledićemo promenljivu `postContent` u svojstvo value klase `textarea` da bismo imali takozvanu **kontrolisanu komponentu**.

Kreiraćemo praznu promenljivu znakovnog niza u inicijalizatoru svojstva state na sledeći način:

```
state = {
  posts: posts,
  postContent: ''
}
```

Zatim ćemo ekstrahovati ovo iz stanja klase unutar metoda `render`:

```
const { posts, postContent } = this.state;
```

Sada nova promenljiva stanja ostaje prazna, mada možemo unutar nje da napišemo textarea. Ovaj problem se dešava zato što direktno menjamo DOM element, ali ne vezujemo događaj promene sa postojećom React funkcijom. Ova funkcija ima zadatak da ažurira React interno stanje koje nije automatski povezano sa DOM stanjem pretraživača.

- U prethodnom kodu smo već prosledili funkciju ažuriranja pod nazivom `this.handlePostContentChange` u svojstvo `onChange` klase `textarea`.

Logički korak je da implementiramo ovu funkciju:

```
handlePostContentChange = (event) => {
  this.setState({postContent: event.target.value})
}
```

Možda ste navikli da pišete ovaj kod malo drugačije:

```
handlePostContentChange(event) {
  this.setState({postContent: event.target.value})
}
```

Ove dve varijante se umnogome razlikuju. Isprobajte sami.

Kada koristimo drugu varijantu, izvršavanje funkcije će dovesti do greške. Oblast važenja unutar funkcije će biti pogrešna i nećemo imati pristup klasi pomoću komande this.

U ovom slučaju je potrebno da napišemo konstruktor za klasu i ručno vežemo oblast važenja sa funkcijom na sledeći način:

```
this.handlePostContentChange = this.handlePostContentChange.bind(this);
```

Veoma lako ćemo imati dodatnih pet linija koda kada pišemo konstruktor za pravilno vezivanje oblasti važenja.

U prvoj varijanti koristi se ES6 arrow funkcija, koja vodi računa o odgovarajućoj oblasti važenja. Preporučujem ovu varijantu, jer je veoma jasna - možemo uštedeti vreme koje bismo utrošili na analizu i pisanje koda.

Ponovo ćemo pogledati pretraživač. Obrazac postoji, ali nije lep, pa ćemo dodati sledeći CSS:

```
form {
  padding-bottom: 20px;
}
form textarea {
  width: calc(100% - 20px);
  padding: 10px;
  border-color: #bbb;
}
form [type=submit] {
  border: none;
  background-color: #6ca6fd;
  color: #fff;
  padding: 10px;
  border-radius: 5px;
  font-size: 14px;
  float: right;
}
```

Poslednji korak je da implementiramo funkciju handleSubmit za obrazac:

```
handleSubmit = (event) => {
  event.preventDefault();
  const newPost = {
    id: this.state.posts.length + 1,
    text: this.state.postContent,
    user: {
      avatar: '/uploads/avatar1.png',
      username: 'Fake User'
    }
  };
  this.setState((prevState) => ({
    posts: [newPost, ...prevState.posts],
    postContent: ''
  }));
}
```

Prethodni kod izgleda komplikovanije nego što jeste, ali ću vam ga brzo objasniti.

Treba da pokrenemo `event.preventDefault` da bismo onemogućili pretraživač da šalje obrazac i da ponovo učitava stranicu (većina ljudi koji se prebacuju sa jQueryja na druge JavaScript radne okvire će znati ovo).

Zatim ćemo snimiti novi post u promenljivu `newPost` koju želimo da dodamo u izvor vesti.

Mi ćemo uneti neke lažne podatke da bismo simulirali aplikaciju iz stvarnog sveta. Za testiranje id novog posta je broj postova u promenljivoj state plus jedan. React želi da dodelimo svakom podređenom elementu u ReactDOM-u jedinstveni id. Brojanjem postova mi simuliramo ponašanje svarne pozadinske komponente koja nam daje jedinstvene id-je za postove.

Tekst za novi post dolazi iz promenljive `postContent` iz stanja komponente. Štaviše, mi još uvek nemamo korisnički sistem, koji GraphQL server može da upotrebi i da nam obezbedi najnovije postove, uključujući i usklađivanje korisnika sa njihovim avatarima. Simuliraćemo ovaj zadatak postavljanjem statičkog objekta korisnika za sve nove postove koje kreiramo.

Na kraju ćemo ažurirati ponovo stanje komponente. Ovde zadatak postaje malo komplikovaniji. Nećemo proslediti objekat kao što radimo unutar funkcije `handlePostContentChange`; prosledićemo funkciju update.

Ovaj pristup nam pouzdano obezbeđuje aktuelno stanje. Generalno, preporučujem upotrebu funkcije, umesto samo objekta - ona vas automatski štiti od problema stanja utrkivanja (kada se dva procesa takmiče za resurs), gde više funkcija manipuliše stanjem. Uvek imajte na umu da je funkcija setState `asinhrona`.

Vraćena vrednost funkcije je objekat stanja koji bismo obično upotrebili direktno. Zahvaljujući ES6 operatoru proširivanja, možemo da dodamo promenljivu `newPost` ispred starih postova, što će renderovati najnoviji post na vrhu liste. Objekat `textarea` je izbrisan prosleđivanjem praznog znakovnog niza u promenljivu `setState` za polje `postContent`.

Sada se slobodno malo poigrajte radnim React obrascem. Nemojte da zaboravite da neće biti trajni svi postovi koje kreirate, jer se oni čuvaju samo u lokalnoj memoriji pretraživača i nisu snimljeni u bazu podataka. Zbog toga će „osvežavanje“ prikaza izbrisati postove.

Kontrolisanje zaglavlja dokumenta pomoću React Helmeta

Kada razvijate veb aplikaciju, važno je da možete da kontrolišete zaglavlje dokumenta. Možda ćete želeti da promenite naslov ili opis na osnovu sadržaja koji predstavljate.

React Helmet je odličan paket koji to obezbeđuje u toku rada, uključujući i redefinisanje više zaglavlja i renderovanje na strani servera.

Instalirajte React Helmet pomoću sledeće komande:

```
npm install --save react-helmet
```

Možete da dodate sva standardna HTML zaglavlja uz React Helmet.

Preporučujem da zadržite standardne oznake head unutar šablona. One imaju prednost zato što, pre nego što je React renderovan, uvek postoji standardno zaglavlje dokumenta. U našem primeru možete direktno da primenite naslov i opis u App.js.

Importujte react-helmet na početak fajla:

```
import { Helmet } from 'react-helmet';
```

Dodajte sam `Helmet` direktno iznad linije `postForm div`:

```
<Helmet>
  <title>Graphbook - Feed</title>
  <meta name="description" content="Newsfeed of all your friends on
    Graphbook" />
</Helmet>
```

Ako ponovo učitate pretraživač i pogledate naslov u traci naslova pretraživača, videćete da se promenio sa `Graphbook` na `Graphbook - Feed`, zato što smo već definisali naslov unutar fajla `index.html`. Kada React završi renderovanje, novo zaglavlje dokumenta je primenjeno.

Produkciona verzija sa webpackom

Poslednji korak za React postavku je podešavanje produkcione verzije. Do sada smo koristili `webpack-dev-server`, ali on izvorno uključuje nepoboljšanu razvojnu verziju. Štaviše, webpack automatski proširuje veb server. U sledećem poglavlju ćemo predstaviti Express.js kao veb server, pa nam neće biti potreban webpack za hostovanje.

Produkciona verzija spaja sve JavaScript fajlove, ali i CSS fajlove u dva posebna fajla, koji mogu da se upotrebe direktno u pretraživaču. Da bismo upakovali CSS fajlove, oslonićemo se na još jedan webpack dodatni modul, koji se zove `MiniCss`:

```
npm install --save-dev mini-css-extract-plugin
```

Ne želimo da promenimo aktuelni `webpack.client.config.js` fajl, jer je kreiran za razvoj. U objekat `scripts` fajla `package.json` dodaćemo sledeću komandu:

```
"client:build": "webpack --config webpack.client.build.config.js"
```

Ova komanda će pokrenuti webpack pomoću individualnog produkcionog webpack konfiguracionog fajla, a sada ćemo kreirati još jedan. Prvo ćemo klonirati originalni fajl `webpack.client.config.js` i promenićemo mu naziv na `webpack.client.build.config.js`.

Promenite sledeće stavke u novom fajlu:

1. Opcija `mode` treba da bude `production`, a ne `development`.
2. Poslaćemo zahtev za `MiniCss` dodatni modul:

```
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
```

3. Zamenićemo aktuelno CSS pravilo:

```
{
  test: /\.css$/,
  use: [{ loader: MiniCssExtractPlugin.loader,
    options: {
      publicPath: '../'
    }
  }, 'css-loader'],
},
```

   Ne koristimo više `style-loader`, već `MiniCss` dodatni modul. Dodatni modul prolazi kroz ceo CSS kod, spaja ga u poseban fajl i uklanja iskaze `import` iz fajla `bundle.js` koji smo istovremeno generisali.

4. Na kraju ćemo dodatni modul priključiti dodatnim modulima na kraju konfiguracionog fajla:

```
new MiniCssExtractPlugin({
  filename: 'bundle.css',

})
```

5. Uklonićemo celo svojstvo .

Kada pokrećemo novu konfiguraciju, ona neće razdeliti server ili prozor pretraživača; samo će kreirati produkcioni JavaScript i CSS paket, koje zahteva u `index.html` fajlu. U skladu sa fajlom `webpack.client.build.config.js`, ova tri fajla će biti snimljena u direktorijum `dist/client`.

Možete da pokrenete ovu komandu izvršavanjem koda npm `run client:build`.

Pogledajte u direktorijum `dist/client` i videćete tri fajla. Možete da otvorite fajl `index.html` u pretraživaču. Nažalost, slike su prekinute, jer URL-ovi slika više nisu tačni. Prihvatićemo ovu grešku na trenutak, jer će ona biti automatski ispravljena kada imamo radnu pozadinsku komponentu.

Sada smo završili osnovnu postavku Reacta.

KORISNE RAZVOJNE ALATKE

Kada koristimo React, želećemo da znamo zašto je aplikacija renderovana na određeni način. Potrebno je da znamo koja svojstva komponenta prima i kako izgleda njen aktuelni status. Pošto ovo nije prikazano u DOM-u ili negde drugde u Chrome DevToolsu, potreban nam je poseban dodatni modul.

„Facebook“ je mislio i na to. Posetite stranicu https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi i instalirajte React Developer Tools. Ovaj dodatni modul omogućava vam da istražite React aplikacije i komponente. Kada ponovo otvorite Chrome DevTools, videćete da postoji nova kartica na kraju reda.

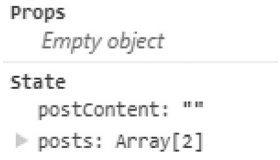
Ako ne možete da vidite ovu karticu, treba da restartujete Chrome. Takođe ćete pronaći i React Developer Tools za Firefox.

Ovaj dodatni modul omogućava da prikažete, pretražite i editujete sve komponente ReactDOM-a.

Panel sa leve strane izgleda kao regularno DOM stablo (Elements) u Chrome DevToolsu, ali, umesto prikaza HTML oznaka, videćete sve komponente koje ste upotrebili unutar stabla. ReactDOM renderuje ovo stablo u stvarni HTML.

Prva komponenta u aktuelnoj verziji Graphbooka treba da bude <App />.

Ako kliknete na komponentu, panel sa desne strane će prikazati svojstva, stanje i kontekst. Možete da isprobate ovo, koristeći komponentu App, koja je jedina prava React komponenta.



Klasa `App` je prva komponenta aplikacije, pa, zbog toga, ne prima svojstva. Podređene klase primaju svojstva iz svojih matičnih klasa; bez matične klase nema svojstava.

Sada testirajte klasu `App` i poigrajte se stanjem. Videćete da menjanje stanja renderuje ReactDOM i ažurira HTML. Možete da editujete promenljivu `postContent`, koja ubacuje novi tekst unutar objekta `textarea`. Kao što možete da vidite, svi događaji su generisani i funkcije za obradu su pokrenute. Ažuriranje stanja uvek pokreće funkciju renderovanja, pa nastojte da ažurirate status što ređe.

Analiza veličine paketa

Ljudi koji pokušavaju da koriste što manje propusnog opsega će želeti da zadrže ograničenu veličinu paketa. Ja preporučujem da uvek pratite veličinu paketa, posebno kada zahtevate više modula pomoću npm-a. U ovom primeru paket brzo može dostići ogromnu veličinu, jer npm paketi često zahtevaju druge npm pakete.

Da biste sprečili povećanje paketa, potreban je metod za analizu veličine paketa. Vredi proveravati samo produkcionu verziju. Kao što je prethodno pomenuto, razvojna verzija, između ostalog, uključuje React u razvojnom izdanju sa mapama izvora.

Zahvaljujući webpacku, postoje jednostavna rešenja za analizu paketa, kao što je `webpack-bundle-analyzer`.

Instalirajte ovaj paket pomoću sledeće komande:

```
npm install --save-dev webpack-bundle-analyzer
```

Zatim je potrebno da dodate dve komande u objekat scripts u fajlu package.json:

- `"stats" - "webpack --profile --json --config webpack.client.build.config.js > stats.json"`
- `"analyze" - "webpack-bundle-analyzer stats.json"`

Prva komanda kreira produkcionu verziju kao i fajl `stats.json` u root direktorijumu. U ovom fajlu se nalaze informacije koje su nam potrebne.

Komanda `analyze` pokreće `webpack-bundle-analyzer` i analizira kako je izgrađen paket i koliko je veliki svaki paket koji koristimo.

Analizu ćemo izvršiti na sledeći način:

```
npm run stats
npm run analyze
```

Možemo da vidimo paket i veličine paketa. Uklonićemo nepotrebne pakete u projektima i videti kako je paket reorganizovan. Možete da vidite primer na sledećoj slici.

Dijagram izgleda kao WinDirStat, koji je softver za prikaz iskorišćenosti diska računara. Možemo da identifikujemo pakete koji čine većinu paketa.

REZIME

U ovom poglavlju smo završili radnu postavku Reacta. Ovo je dobra početna tačka za čeoni prikaz. Možemo da napišemo i izgradimo statičke veb stranice pomoću ove postavke.

U sledećem poglavlju ćemo se primarno fokusirati na podešavanje za pozadinsku komponentu. Konfigurisaćemo Express.js da prihvati prve zahteve i prosledi sve GraphQL upite u Apollo. Takođe ćete naučiti kako da upotrebite Postman za testiranje API-a.