React i React Native

prevod IV izdanja

Izgradnja međuplatformskih JavaScript aplikacija za veb, desktop i mobilne uređaje

Adam Boduch | Roy Derks | Mikhail Sakhniuk

kompjuter biblioteka

React i React Native

prevod IV izdanja

Skenirajte QR kod, registrujte knjigu i osvojite nagradu

Tokom niza godina, React i React Native su se pokazali među JavaScript programerima kao popularan izbor za kompletan i praktičan vodič za React eko-sistem. Ovo četvrto izdanje sadrži najnovije funkcije, poboljšanja i ispravke kako bi se uskladilo sa Reactom 18, a istovremeno je kompatibilno i sa React Nativeom. Uključuje nova poglavlja kojima su obuhvaćene važne karakteristike i koncepti u savremenom razvoju aplikacija na više platformi pomoću Reacta.

Ovaj definitivni vodič, u kojem su opisane osnove Reacta i popularne komponente, kao što su Hooks, GraphQL i NativeBase, pomoći će vam da postanete „korak po korak“ profesionalni React programer.

Prvo ćete učiti o osnovnim gradivnim blokovima React komponenata. Dok budete čitali poglavlja, koristićete funkcionalnosti višeg nivoa u razvoju aplikacija, a zatim ćete primeniti svoje znanje u praksi, tako što ćete razvijati komponente korisničkog interfejsa za veb i nativne platforme. U završnim poglavljima ćete naučiti kako da svoju aplikaciju povežete sa robusnom arhitekturom podataka.

Kada pročitate celu knjigu, moći ćete samouvereno da kreirate React aplikacije za veb i React Native aplikacije za više platformi – na primer, za veb, mobilne i desktop platforme.

Šta ćete naučiti:

- da istražite React arhitekturu, svojstva komponenata, stanje i Context
- da koristite React Hooks za rukovanje funkcijama i komponentama
- da implementirate podelu koda, koristeći „lenje“ komponente i Suspense
- da izradite robustan korisnički interfejs za mobilne i desktop aplikacije, koristeći Material-UI
- da pišite zajedničke komponente za Android i iOS aplikacije, koristeći React Native
- da pojednostavite definisanje rasporeda elemenata na stranici (layout) za React Native aplikacije, koristeći NativeBase
- da pišete GraphQL šeme za osnaživanje veb i mobilnih aplikacija
- da implementirate komponente vođene Apollom

ISBN: 978-86-7310-584-0

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: Adam Boduch
Roy Derks
Mikhail Sakhniuk

Prevod: Biljana Tešić

Lektura: Miloš Jevtović

Slog: Zvonko Aleksić

Znak Kompjuter biblioteke:
Miloš Milosavljević

Štampa: „Pekograf“, Zemun

Tiraž: 500

Godina izdanja: 2023.

Broj knjige: 561

Izdanje: Prvo

ISBN: 978-86-7310-584-0

React and React Native - Fourth Edition

Adam Boduch
Roy Derks
Mikhail Sakhniuk

ISBN 978-1-80323-128-0

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд

004.42:004.738.1
004.438JAVASCRIPT
004.42REACT

БОДУЧ, Адам
React i React Native : prevod IV izdanja : izgradnja međuplatformskih JavaScript aplikacija za veb, desktop i mobilne uređaje / Adam Boduch, Roy Derks, Mikhail Sakhniuk; [prevod Biljana Tešić]. - 1. izd. - Beograd: Kompjuter Biblioteka, 2023 (Zemun : Pekograf). - XXV, 575 str.: ilustr.; 26 cm. - (Kompjuter biblioteka; br. knj. 561)

Prevod dela: React and React Native. - Tiraž 500. -
O autorima: str. III. - Registar.

ISBN 978-86-7310-584-0

1. Деркс, Роj [аутор] 2. Сахнюк, Михаил [аутор]
а) Web презентације -- Програмирање
б) Програмски језик „JavaScript“
в) Апликативни програм „React“

COBISS.SR-ID 85181193COBISS.SR-ID 75527177

O AUTORIMA

Adam Boduch je skoro 15 godina uključen u razvoj JavaScripta velikih razmera. Pre nego što je prešao na frontend, izradio je nekoliko proizvoda velikih razmera za računarstvo u „oblaku“, koristeći Python i Linux. Adam može da se „nosi“ sa problemima i izazovima, jer ima praktično iskustvo u stvarnim softverskim sistemima i izazovima skaliranja koje oni predstavljaju.

Hvala React timu, jer je vebu obezbedio ovu fantastičnu alatku.

Roy Derks je serijski start-up tehnički direktor, međunarodni govornik i autor knjiga iz Holandije. Koristi React, React Native i GraphQL od 2016. godine. Možda ga poznajete kao autora knjige „React Projects – Second Edition“, koju je „Packt“ objavio ranije ove godine. Tokom poslednjih nekoliko godina inspirisao je desetine hiljada programera širom sveta svojim govorima, knjigama, radionicama i kursevima.

Mikhail Sakhniuk je softverski inženjer koji je stručnjak za JavaScript, React i React Native. Ima više od šest godina iskustva u razvoju veb i mobilnih aplikacija. Radio je za start-up kompanije, finansijsko-tehnološke kompanije i produkcijske kuće koje imaju više od 30 miliona korisnika. Trenutno radi kao viši frontend inženjer. Osim toga, poseduje i održava niz projekata otvorenog koda. Svoje znanje i iskustvo prenosi i u knjigama i člancima.

Zahvaljujem se celoj JavaScript zajednici na kreiranju, održavanju i korišćenju hiljade biblioteka i alatki koje čine veb bržim, većim i pristupačnijim

O RECENZENTIMA

Kirill Ezhemenskii je iskusni softverski inženjer, frontend i mobilni programer, stručnjak za razvoj poslovnih rešenja i tehnički direktor u jednoj zdravstvenoj kompaniji. On je zagovornik funkcionalnog programiranja i stručnjak za React Stack, GraphQL i TypeScript. Kirill je takođe React Native mentor.

Sunki Baek je iskusni frontend programer, koji prvenstveno koristi React Native za razvoj aplikacija za više platformi. Njegovo React Native „putovanje“ počelo je 2016. godine, ubrzo nakon što je React Native objavljen, a njegov prvi projekat je bio kreiranje aplikacije za e-trgovinu sa funkcijom ćaskanja. Od onda je uključen u e-trgovinu, online-oflajn trgovinu i projekte prehrambene tehnologije. Trenutno radi kao viši programer mobilnih uređaja u timu Restaurant Live Order u kompaniji „SkipThe Dishes“.

Andrew Baisden je softverski programer koji ima iskustvo u radu na različitim tehničkim stekovima. Pre svega, vešt je kao JavaScript programer, a takođe su mu poznati programski jezici Python i C#. Ima fakultetsku diplomu i proveo je mnogo vremena za proširivanje svog znanja i svojih veština. Tehničko pisanje i kreiranje sadržaja su oblasti u kojima se takođe ističe. Mnogo njegovih članaka se deli na društvenim mrežama i koristi se kao motivacija i osnovni resursi za programere koji pokušavaju da se „probiju“ u softverskoj industriji. Sa sve većom publikom koja broji više od 5.000 članova, Endru nastavlja da daje svoj doprinos gde god da krene.

Kratak sadržaj

Sadržaj

POGLAVLJE 7

POGLAVLJE 8

POGLAVLJE 9

POGLAVLJE 19

POGLAVLJE 20

POGLAVLJE 21

POGLAVLJE 22

POGLAVLJE 23

POGLAVLJE 24

POGLAVLJE 25

POGLAVLJE 26

POGLAVLJE 27

POGLAVLJE 28

DEO 3

POGLAVLJE 29

POGLAVLJE 30

POGLAVLJE 31

Predgovor

Tokom godina su se React i React Native pokazali među JavaScript programerima kao popularan izbor za kompletan i praktičan vodič za React eko-sistem. Ovo četvorto izdanje sadrži najnovije funkcije, poboljšanja i ispravke kako bi se uskladilo sa Reactom 18, a istovremeno je kompatibilno i sa React Nativeom. Uključuje nova poglavlja kojima su obuhvaćene važne karakteristike i koncepte u savremenom razvoju aplikacija na više platformi pomoću Reacta.

Ovaj definitivni vodič, u kojem su opisane osnove Reacta i popularne komponente, kao što su Hooks, GraphQL i NativeBase, pomoći će vam da, „korak po korak", postanete profesionalni React programer.

Prvo ćete učiti o osnovnim gradivnim blokovima React komponenata. Dok budete čitali poglavlja, koristićete funkcionalnosti višeg nivoa u razvoju aplikacija, a zatim ćete primeniti svoje znanje u praksi, tako što ćete razvijati komponente korisničkog interfejsa za veb i nativne platforme. U završnim poglavljima ćete naučiti kako da svoju aplikaciju povežete sa robusnom arhitekturom podataka.

Kada pročitate celu knjigu, moći ćete samouvereno da kreirate React aplikacije za veb i React Native aplikacije za više platformi – na primer, za veb, mobilne i desktop platforme.

Za koga je ova knjiga?

Ova knjiga je za svakog JavaScript programera koji želi da počne da uči kako da koristi React i React Native za razvoj mobilnih i veb aplikacija. Nije potrebno prethodno poznavanje Reacta; međutim, potrebno je osnovno poznavanje JavaScripta da biste mogli da pratite sadržaj ove knjige.

Šta obuhvata ova knjiga?

U Poglavlju 1, „Zašto React?", opisano je šta je React i zašto treba da ga koristite za kreiranje aplikacije.

U Poglavlju 2, „Renderovanje pomoću JSX-a“, naučićete osnove JSX-a, jezika za označavanje koji koriste React komponente.

U Poglavlju 3, „Komponente svojstava, stanje i Context“, predstavljani su osnovni mehanizmi prosleđivanja podataka u React aplikaciji.

U Poglavlju 4, „Početak rada sa Hooksima“, pokazano je kako se React Hooks mogu koristiti za proširenje ponašanja komponenata.

Poglavlje 5, „Rukovanje događajima na React način“, prikazan je način na koji događajima rukuju React komponente.

Poglavlje 6, „Izrada komponenata za višekratnu upotrebu“, vodi vas kroz proces refaktorisanja komponenata u primeru.

U poglavlju 7, „‚Životni ciklus‘ React komponenata“, opisane su različite faze kroz koje prolaze React komponente i zašto su one važne za React programere.

U Poglavlju 8, „Provera svojstava komponenata“, prikazano je kako možete da obezbedite da vrednosti svojstava React komponente budu očekivane.

Poglavlje 9, „Rukovanje navigacijom sa rutama“, sadrži mnoštvo primera kako da podesite rutiranje za vašu React veb aplikaciju.

U Poglavlju 10, „Deljenje koda korišćenjem ‚lenjih‘ komponenata i Suspensea“, predstavljene su tehnike za deljenje koda koje rezultiraju manjim, efikasnijim aplikacijama.

U Poglavlju 11, „React komponente na strani servera“, naučićete kako da koristite Next.js za kreiranje velikih React aplikacija koje prikazuju sadržaj na serveru i klijentu.

U Poglavlju 12, „Komponente radnog okvira korisničkog interfejsa“, prikazano je kako da započnete rad u MUI-u, React biblioteci komponenata za izradu korisničkog interfejsa (UI-a).

U Poglavlju 13, „Ažuriranja stanja visokih performansi“, biće više reči o novim funkcijama u Reactu 18 koje omogućavaju efikasna ažuriranja stanja i aplikaciju visokih performansi.

U Poglavlju 14, „Zašto React Native?“, opisano je šta je React Native biblioteka i koje su razlike između nativnih mobilnih programera.

Poglavlje 15, „React Native ispod ‚haube‘“, sadrži pregled arhitekture React Nativea.

U Poglavlju 16, „Započnite React Native projekte“, naučićete kako da započnete novi React Native projekat.

U Poglavlju 17, „Izrada prilagodljivih rasporeda elemenata pomoću Flexboxa“, opisano je kako možete da kreirate raspored elemenata i dodate stilove.

U Poglavlju 18, „Navigacija između ekrana“, prikazani su pristupi za prebacivanje sa ekrana na ekran u aplikaciji.

U Poglavlju 19, „Renderovanje lista stavki“, opisano je kako možete da implementirate liste podataka u aplikaciju.

U Poglavlju 20, „Prikazivanje toka“, naučićete kako da rukujete indikacijama procesa i trakama toka.

U Poglavlju 21, „Geolokacija i mape“, prikazano je kako da pratite geolokaciju i dodate mapu u aplikaciju.

Poglavlje 22, „Prikupljanje korisničkog unosa“, naučićete kako da kreirate obrasce.

U Poglavlju 23, „Prikazivanje modalnih ekrana“, naučićete kako da kreirate modalne okvire za dijalog.

Poglavlje 24, „Odgovaranje na korisničke gestove“, sadrži primere za rukovanje korisničkim gestovima.

U Poglavlju 25, „Korišćenje animacija“, opisano je kako možete da implementirate animacije u aplikaciju.

U Poglavlju 26, „Kontrolisanje prikaza slike“, pokazano je kako možete da prikažete slike u aplikaciji React Native.

U Poglavlju 27, „Isključivanje sa mreže“, prikazano je kako možete da se „nosite“ sa aplikacijom kada mobilni telefon nema internet vezu.

U Poglavlju 28, „Izbor komponenata nativnog korisničkog interfejsa pomoću NativeBasea“, naučićete kako da kreirate aplikaciju koristeći NativeBase UI biblioteku.

U Poglavlju 29, „Rukovanje stanjem aplikacije“, saznaćete kako možete da rukujete stanjem aplikacije i za veb i za mobilne aplikacije.

U Poglavlju 30, „Zašto GraphQL?“, opisano je šta je GraphQL i kako se koristi.

U Poglavlju 31, „Izrada React GraphQL aplikacije“, pokazano je kako se rukuje GrapQL-om u React i React Native aplikacijama.

Izvucite maksimum iz ove knjige

Pretpostavljamo da imate osnovno znanje o programskom jeziku *programskog jezika JavaScript. Takođe pretpostavljamo da ćete pratiti primere koji zahtevaju terminal komandne linije, uređivač koda i veb pregledač.*

Zahtevi za učenje React Nativea su isti kao i za razvoj Reacta, ali da biste pokrenuli aplikaciju na stvarnom uređaju, biće vam potreban Android ili iOS pametni telefon. Da biste pokrenuli iOS aplikacije u simulatoru, biće vam potreban Mac računar.

Softveri/hardveri obuhvaćeni knjigom	Zahtevi operativnog sistema
React React Native	Windows, macOS ili Linux

Svako poglavlje ima svoju fasciklu u spremištu koda, a svaki primer funkcioniše nezavisno od ostalih. Uopšteno govoreći, možete koristiti `npm install` *i* `npm start` *za pokretanje svakog primera. U datotekama* `README` *u svakoj fascikli pogledajte detaljnija uputstva koja se odnose na svaki konkretan primer.*

Ako koristite digitalnu verziju ove knjige, savetujemo vam da sami unesete kod ili pristupite kodu iz GitHub spremišta knjige (link se nalazi u sledećem odeljku) da biste izbegli sve moguće greške prilikom kopiranja i „lepljenja" koda.

Preuzimanje datoteka primera koda

Možete da preuzmete datoteke sa primerima koda za ovu knjigu sa GitHuba https://github.com/PacktPublishing/React-and-React-Native-4th-Edition. Kod, ako postoji njegovo ažuriranje, biće ažuriran u GitHub spremištu.

Na raspolaganju su vam i drugi paketi kodova iz našeg bogatog kataloga knjiga i video-zapisa dostupnih na adresi https://github.com/PacktPublishing/.

Preuzmite slike u boji za ovu knjigu

Takođe smo vam obezbedili PDF datoteku koja sadrži slike u boji ekrana/dijagrama koji su upotrebljeni u ovoj knjizi. Tu datoteku možete da preuzmete sa adrese https://static.packt-cdn.com/downloads/9781803231280_ColorImages.pdf.

Korišćene konvencije

U ovoj knjizi se koristi niz konvencija.

Kod u tekstu - Označava kodne reči u tekstu, nazive tabela baza podataka, nazive direktorijuma, nazive datoteka, ekstenzije datoteka, nazive putanja, lažne URL adrese, korisnički unos i Twitter postove. Evo primera: „Imate stvarne rute koje su deklarisane kao `<Route>` elementi."

Blok koda je postavljen na sledeći način:

```
  export default function First()
    { return <p>Feature 1, page 1</p>;
}
```

Kada želimo da vam skrenemo pažnju na određeni deo bloka koda, relevantne linije ili stavke će biti prikazane podebljanim slovima:

```
  export default function List({ data, fetchItems, refreshItems,
  isRefreshing }) {
    return (
      <FlatList
        data={data}
        renderItem={({ item }) => <Text style={styles.
  item}>{item.value}</Text>}
```

```
            onEndReached={fetchItems}
            onRefresh={refreshItems}
            refreshing={isRefreshing}
        />
    );
}
```

Svi unosi ili ispisi komandne linije napisani su na sledeći način:

```
npm install @react-navigation/bottom-tabs @react-navigation/
drawer
```

Podebljana slova - Novi termini, važne reči ili reči koje vidite na ekranu - na primer, u menijima ili okvirima za dijalog, biće prikazani u tekstu **podebljanim slovima**. Na primer: „**Komponenta kontejnera (Container Component)** obično sadrži jedan direktni podređeni element.“

Napomene se prikazuju ovako.

Saveti se prikazuju ovako.

Stupite u kontakt

Povratne informacije naših čitalaca su uvek dobrodošle.

Opšte povratne informacije - Ako imate pitanja o bilo kojem aspektu ove knjige, navedite njen naslov u temi vaše poruke i pošaljite nam e-mail na adresu customercare@packtpub.com

Štamparske greške - Iako smo preduzeli sve mere da bismo obezbedili tačnost sadržaja, greške su moguće. Ako pronađete neku grešku u ovoj knjizi, bili bismo zahvalni ako biste nam to javili. Otvorite stranicu http://www.packtpub.com/support/errata i popunite obrazac.

Piraterija - Ako na Internetu pronađete ilegalne kopije naših knjiga, u bilo kojoj formi, molimo vas da nas o tome obavestite i da nam pošaljete adresu lokacije ili naziv veb sajta. Pošaljite nam poruku na adresu copyright@packt.com i pošaljite nam link ka sumnjivom materijalu.

Ako ste zainteresovani da postanete autor - Ako postoji tema za koju ste stručni, a zainteresovani ste za pisanje ili doprinos knjizi, posetite stranicu authors.packtpub.com.

Postanite član Kompjuter biblioteke

Kupovinom jedne naše knjige stekli ste pravo da postanete član Kompjuter biblioteke. Kao član možete da kupujete knjige u pretplati sa 40% popustai učestvujete u akcijama kada ostvarujete popuste na sva naša izdanja. Potrebno je samo da se prijavite preko formulara na našem sajtu. Link za prijavu: http://bit.ly/2TxekSa

Skenirajte QR kod
registrujte knjigu
i osvojite nagradu

DEO 1

React

U ovom delu ćemo razmotriti osnove React alatki i koncepata, primenjujući ih za kreiranje veb aplikacija visokih performansi.

Ovaj deo sadrži sledeća poglavlja:

- Poglavlje 1, „Zašto React?“
- Poglavlje 2, „Renderovanje pomoću JSX-a“
- Poglavlje 3, „Komponente svojstava, stanje i Context“
- Poglavlje 4, „Početak rada u Hooksima“
- Poglavlje 5, „Rukovanje događajima na React način“
- Poglavlje 6, „Izrada komponenata za višekratnu upotrebu“
- Poglavlje 7, „‚Životni ciklus‘ React komponenata“
- Poglavlje 8, „Provera svojstava komponenata“
- Poglavlje 9, „Rukovanje navigacijom sa rutama“
- Poglavlje 10, „Deljenje koda korišćenjem ‚lenjih‘ komponenata i Suspensea“
- Poglavlje 11, „React komponente na strani servera“
- Poglavlje 12, „Komponente radnog okvira korisničkog interfejsa“
- Poglavlje 13, „Ažuriranja stanja visokih performansi“

1

Zašto React?

Ako čitate ovu knjigu, verovatno znate šta je React. Ako ne znate, ne brinite. Daću sve od sebe da filozofske definicije svedem na minimum. Međutim, ovo je dugačka knjiga sa mnogo sadržaja, tako da smatram da je određivanje ritma učenja prvi odgovarajući korak. Da, cilj je naučiti React i React Native. Međutim, u to spada i sastavljanje trajne arhitekture koja može da se pobrine za sve što želimo da izradimo pomoću Reacta danas i u budućnosti.

Ovo poglavlje počinje kratkim objašnjenjem zašto React postoji. Zatim ćemo razmisliti o jednostavnosti Reacta i o tome kako on može da reši mnoge uobičajene probleme performansi sa kojima se suočavaju veb programeri. Nakon toga ćemo razmotriti deklarativne filozofije Reacta i nivo apstrakcije koji React programeri mogu očekivati da će koristiti. Na kraju, dotaknućemo se nekih od glavnih karakteristika Reacta.

Kada budete imali konceptualno znanje Reacta i načina na koji on rešava probleme razvoja korisničkog interfejsa, moći ćete bolje da se pozabavite ostatkom knjige. Ovim poglavljem su obuhvaćene sledeće teme:

- Šta je React?
- React funkcije
- Šta je novo u Reactu 18?

Šta je React?

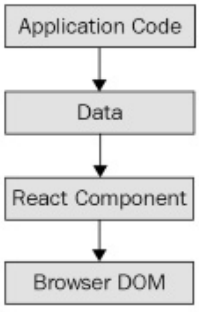
Mislim da je opis **Reacta** u jednom redu na njegovoj početnoj stranici (https://reactjs.org/) koncizan i tačan:

„React je JavaScript biblioteka za kreiranje korisničkih interfejsa."

On je biblioteka za kreiranje **korisničkih interfejsa (UI-a)**. To je savršeno, jer, kako se ispostavilo, sve što uglavnom želimo da radimo u Reactu je da kreiramo korisničke interfejse. Mislim da je najbolji deo ovog opisa sve ono što je izostavljeno. React nije mega radni okvir. Nije full-stack rešenje koje će se pobrinuti za sve, od baze podataka do ažuriranja u realnom vremenu, preko WebSocket veza. Možda zapravo ne želimo većinu ovih unapred upakovanih rešenja. Ako React nije radni okvir, šta je onda tačno?

React je samo sloj prikaza

React se generalno smatra slojem prikaza u aplikaciji. Možda ste ranije koristili biblioteke kao što su Handlebars ili jQuery. Baš kao što jQuery manipuliše elementima korisničkog interfejsa, a Handlebars šabloni se „ubacuju" u stranicu, React komponente menjaju ono što korisnik vidi. Na sledećem dijagramu je prikazano gde se React uklapa u naš frontend kod.



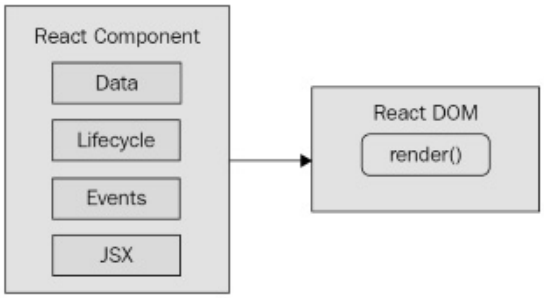
Slika 1.1 Slojevi React aplikacije

To je sve što React može ponuditi, tj. osnovni koncept. Naravno, biće suptilnih varijacija ove teme dok budete čitali knjigu, ali tok je manje-više isti. Imamo neku logiku aplikacije koja generiše neke podatke. Želimo da prikažemo ove podatke korisničkom interfejsu, pa ih prosleđujemo React komponenti, koja rukuje unošenjem HTML-a na stranicu.

Možda se pitate zašto je React toliko važan; izgleda da je on još jedna tehnologija renderovanja. U preostalim odeljcima ovog poglavlja ćemo se dotaći nekih ključnih oblasti u kojima React može da pojednostavi razvoj aplikacija.

Jednostavnost je dobra

React nema mnogo neotkrivenih delova za učenje i razumevanje. Na unutrašnjem planu mnogo štošta se dešava, a mi ćemo se dotaći tih „stvari“ u celoj knjizi. Prednost posedovanja malog API-a za rad je da možete provesti više vremena upoznajući ga, eksperimentišući sa njim i slično, dok je u velikim radnim okvirima vaše vreme posvećeno otkrivanju kako sve funkcioniše. Na sledećem dijagramu su u „grubim“ crtama predstavljeni API-i koje morate imati na umu kada programirate pomoću Reacta.



Slika 1.2 Jednostavnost React API-a

React je podeljen na dva glavna API-a:

- **React Component API** - Ovaj API su delovi stranice koje prikazuje React DOM.
- **React DOM** - Ovo je API koji se koristi za izvršavanje renderovanja na veb stranici.

U okviru React komponente se nalaze sledeće oblasti koje treba imati na umu:

- **Data** - Ovo su podaci koji dolaze odnekud (komponenti nije važno odakle) i koje renderuje komponenta.
- **Lifecycle** - Ova oblast se sastoji od metoda ili Hooksa koje implementiramo da bismo odgovorili na faze ulaska i izlaska komponente u React procesu renderovanja, koje se dešavaju tokom vremena – na primer, jedna faza „životnog ciklusa“ označava vreme tokom kojeg se prikazuje komponenta.
- **Events** - Ovo je kod koji pišemo za odgovaranje na interakcije korisnika.
- **JSX** - Ovo je sintaksa React komponenata koja se koristi za opisivanje struktura korisničkog interfejsa.

Nemojte se još uvek fokusirati na to šta ove različite oblasti React API-a predstavljaju. Zaključak je da je React, po prirodi, jednostavan. Pogledajte samo koliko malo čega ima da se otkrije! To znači da ovde ne moramo da trošimo mnogo vremena na razmatranje detalja API-a. Umesto toga, kada shvatite osnove, možete potrošiti više vremena na nijansirane React obrasce korišćenja koji se dobro uklapaju u deklarativne strukture korisničkog interfejsa.

Deklarativne strukture korisničkog interfejsa

Novajlijama u Reactu je teško da se uhvate u koštac sa idejom da se komponente kombinuju u označavanju pomoću JavaScripta kako bi bile deklarisane strukture korisničkog interfejsa. Ako ste pogledali React primere i uočili istu neželjenu reakciju, ne brinite. U početku smo svi skeptični prema ovom pristupu, a mislim da je razlog taj što smo decenijama bili uslovljeni principom razdvajanja nadležnosti, prema kojem različite nadležnosti, kao što su logika i prezentacija, treba da budu odvojene jedna od druge. Sada, kad god vidimo da su neke „stvari" pomešane, automatski pretpostavljamo da je to loše i da to ne bi trebalo da se desi.

Sintaksa koju koriste React komponente naziva se JSX (JavaScript XML). Komponenta prikazuje sadržaj, tako što vraća neki JSX. Sam JSX je obično HTML označavanje, kombinovano sa prilagođenim oznakama za React komponente. Specifičnosti u ovom trenutku nisu važne; „zaronićemo" u detalje u narednim poglavljima. Ono što je revolucionarno u deklarativnom JSX pristupu je da ne moramo da izvodimo male mikrooperacije da bismo promenili sadržaj komponente.

Važna napomena

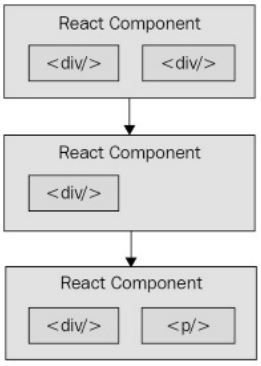
Iako neću slediti konvenciju u ovoj knjizi, neki React programeri preferiraju ekstenziju .jsx, umesto ekstenzije .js, za svoju komponentu.

Na primer, razmislite o korišćenju nečega kao što je jQuery za kreiranje aplikacije. Imate stranicu sa nekim sadržajem i želite da dodate klasu u pasus nakon klika na neko dugme. Izvođenje ovih koraka je veoma jednostavno. To se zove imperativno programiranje i problematično je za razvoj korisničkog interfejsa. Iako je ovaj primer izmene klase elementa jednostavan, stvarne aplikacije obično uključuju više od tri ili četiri koraka za izvršavanje nekih radnji.

React komponente ne zahtevaju da izvršite korake na imperativan način. Zbog toga je JSX centralni deo React komponenata. Sintaksa u XML stilu olakšava opisivanje izgleda korisničkog interfejsa, odnosno HTML elemenata koje će ova komponenta renderovati. To se zove deklarativno programiranje i veoma je pogodno za razvoj korisničkog interfejsa. Kada deklarišete svoju strukturu korisničkog interfejsa, morate da navedete kako se ona menja tokom vremena.

Podaci se menjaju tokom vremena

Još jedna oblast koju teško shvataju novajlije u Reactu je ideja da je JSX poput statičkog znakovnog niza, koji predstavlja deo renderovanog izlaza. Sada „u igru" ulaze vreme i podaci. React komponente se oslanjaju na podatke koji su im prosleđeni. Ovi podaci predstavljaju dinamičke delove korisničkog interfejsa – na primer, element korisničkog interfejsa koji je renderovan na osnovu logičke vrednosti mogao bi da se promeni kada se sledećeg puta komponenta renderuje. Na sledećoj slici je dijagram na kojem je prikazana ta ideja.



Slika 1.3 Komponente reakcije koje se menjaju tokom vremena

Kada se React komponenta renderuje, to je kao da pravite snimak JSX-a u određenom trenutku. Kako se vaša aplikacija kreće napred kroz vreme, tako imate uređenu kolekciju renderovanih komponenata korisničkog interfejsa. Osim što deklarativno opisuje šta bi korisnički inerfejs trebalo da bude, ponovno renderovanje istog JSX sadržaja čini „stvari" mnogo lakšim za programere. Izazov je osigurati da React može da se „nosi" sa zahtevima performansi ovog pristupa.

Performanse su važne

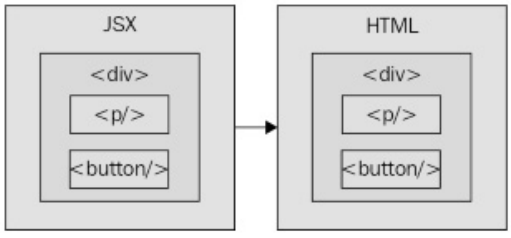
Korišćenje Reacta za kreiranje korisničkih interfejsa označava da možemo deklarisati strukturu korisničkog interfejsa pomoću JSX-a, što je manje podložno greškama od imperativnog pristupa sastavljanja korisničkog interfejsa deo po deo. Međutim, deklarativni pristup predstavlja izazov, tj. performanse.

Na primer, deklarativna struktura korisničkog interfejsa je dobra za početno renderovanje, jer na stranici još nema ničega. Dakle, React renderer može da „vidi" strukturu deklarisanu u JSX-u i da je prikaže u DOM-u pregledača.

Važna napomena

Objektni model dokumenta (DOM - Document Object Model) predstavlja HTML u pregledaču nakon što je renderovan. DOM API služi da JavaScript može da promeni sadržaj na stranici.

Ovaj koncept je ilustrovan na dijagramu na sledećoj slici.

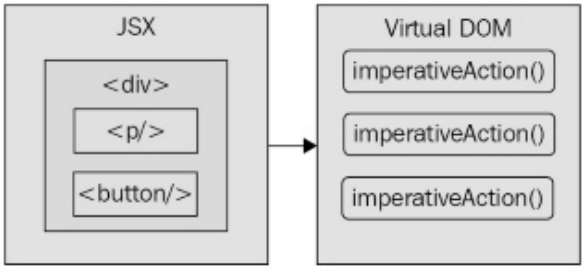


Slika 1.4 Način na koji se JSX sintaksa prevodi u HTML u DOM pregledača

Na početnom renderu React komponente i njihov JSX se ne razlikuju od drugih biblioteka šablona. Na primer, Handlebars će prikazati šablon za HTML označavanje kao znakovni niz, koji će, zatim, biti umetnut u DOM pregledača. React se razlikuje od biblioteka kao što je Handlebars po tome što se podaci menjaju i moramo ponovo da renderujemo komponentu. Handlebars će samo ponovo izraditi ceo HTML znakovni niz na isti način na koji je to uradio na početnom renderu. Pošto je to problematično za performanse, često na kraju implementiramo imperativna zaobilazna rešenja koja ručno ažuriraju male delove DOM-a. Na kraju imamo „metež" deklarativnih šablona i imperativnog koda za rukovanje dinamičkim aspektima korisničkog interfejsa.

U Reactu se navedeni problemi ne mogu desiti. To je ono što ga izdvaja od drugih biblioteka prikaza. Komponente su deklarativne za početno renderovanje i ostaju takve, čak i kada se ponovo renderuju. To je ono što React radi „ispod haube", a što omogućava ponovno renderovanje deklarativnih struktura korisničkog interfejsa.

React ima virtuelni DOM, koji se koristi da zadrži reprezentaciju stvarnih DOM elemenata u memoriji. Uvek kada ponovo renderujemo komponentu, virtuelni DOM može da uporedi novi sadržaj sa sadržajem koji je već renderovan na stranici. Na osnovu razlike, virtuelni DOM može da izvrši imperativne korake neophodne za unošenje izmena. Dakle, ne samo da možemo da zadržimo naš deklarativni kod kada treba da ažuriramo korisnički interfejs, već će se i React pobrinuti da to bude urađeno na efikasan način. Kako ovaj proces izgleda vidi se na sledećoj slici.



Slika 1.5 React prevodi JSX sintaksu u imperativne DOM API pozive

Važna napomena

Kada čitate o Reactu, često ćete naići na reči kao što su razlikovanje i „krpljenje". Razlikovanje znači upoređivanje starog sadržaja sa novim da bi se otkrilo šta se promenilo. „Krpljenje" znači izvršavanje neophodnih DOM operacija za renderovanje novog sadržaja.

Baš kao bilo koja druga JavaScript biblioteka, React je ograničen prirodom glavne programske niti, od pokretanja do završetka. Na primer, ako su unutrašnji elementi Reacta zauzeti razlikovanjem sadržaja i „krpljenjem" DOM-a, pregledač ne može da odgovori na korisnički unos. Kao što ćete videti u poslednjem odeljku ovog poglavlja, izmene su izvršene u internim algoritmima za prikazivanje u Reactu 16 da bi bili ublaženi ovi problemi performansi. Pošto smo rešili probleme performansi, moramo da budemo sigurni da je React dovoljno fleksibilan da se prilagodi različitim platformama na kojima bismo možda želeli da primenimo naše aplikacije u budućnosti.

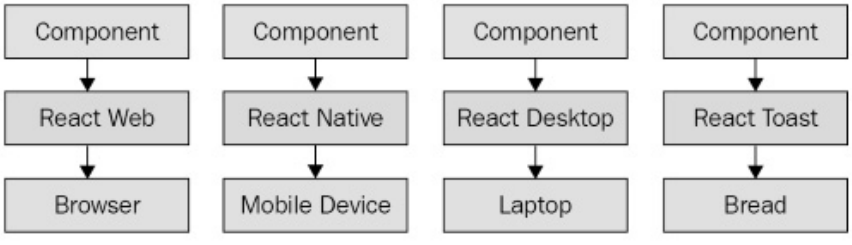
Pravi nivo apstrakcije

Još jedna tema koju želim da detaljno razmotrim pre nego što „zaronimo" u React kod je apstrakcija.

U prethodnom odeljku videli ste kako se JSX sintaksa prevodi u operacije niskog nivoa koje ažuriraju naš korisnički interfejs. Kako React prevodi naše deklarativne komponente korisničkog interfejsa možemo bolje da vidimo ako nam nije mnogo važno šta je cilj renderovanja. Cilj renderovanja je slučajno DOM pregledača sa Reactom, ali nije ograničen na DOM pregledača.

React može da se koristi za bilo koji korisnički interfejs koji želimo da kreiramo, na bilo kojem mogućem uređaju.Tek počinjemo da uviđamo to koristeći React Native, ali mogućnosti su beskrajne. Ja lično neću biti iznenađen ako React Toast postane „stvar", ciljajući tostere koji mogu ispeći renderovani izlaz JSX-a u hleb. Nivo apstrakcije u Reactu je na pravom nivou i na pravom mestu.

Dijagram na sledećoj slici vam daje ideju o tome kako React ne cilja samo pregledač.



Slika 1.6 React apstrahuje ciljno okruženje za renderovanje iz komponenata koje implementiramo

Sleva nadesno imamo **React Web** (samo običan React), **React Native**, **React Desktop** i **React Toast**. Kao što vidite, da biste ciljali nešto novo, primenjuje se isti obrazac:

- implementiranje komponente specifične za cilj
- implementiranje React renderera koji može da izvodi operacije specifične za platformu „ispod haube“

Ovo je, očigledno, preveliko pojednostavljenje onoga što je zapravo implementirano za bilo koje React okruženje. Međutim, detalji nam nisu mnogo važni. Važno je da možemo da koristimo naše React znanje da bismo se fokusirali na opisivanje strukture našeg korisničkog interfejsa na bilo kojoj platformi.

Važna napomena

Nažalost, React Toast verovatno nikada neće postojati.

Sada, kada razumete ulogu apstrakcija u Reactu, objasniću šta je novo u Reactu 18.

Šta je novo u Reactu 18?

Primeri u ovoj knjizi su zasnovani na Reactu 18. U ovoj verziji sveobuhvatne promene API-a nisu uvedene na način na koji je to urađeno u Reactu 16. Postoje, međutim, dve značajne promene koje ćemo detaljnije razmotriti u Poglavlju 13, „Ažuriranja stanja visokih performansi“.

Automatsko grupisanje

Grupisanje ažuriranja stanja drastično poboljšava performanse React aplikacija, jer smanjuje broj rendera koje treba izvršiti. React je oduvek mogao da grupiše više ažuriranja stanja u jedno ažuriranje, ali je bilo ograničeno gde se to moglo dogoditi. Konkretno, mogli biste samo da grupišete ažuriranja stanja unutar hendlera događaja. Ovde je problem da se veći deo našeg koda za ažuriranje stanja izvršava na asinhroni način koji sprečava da se desi automatsko grupisanje.

React 18 uklanja ovu barijeru i omogućava da se automatsko grupisanje ažuriranja stanja izvrši bilo gde. U Poglavlju 13, „Ažuriranja stanja visokih performansi“, videćete primere u kojima se upoređuje kako je grupisanje funkcionisalo pre Reacta 18 i šta možete očekivati sada.

Tranzicije stanja

React 18 uvodi pojam tranzicije stanja. Ideja sa tranzicijama stanja je da manje važna ažuriranja stanja koja se izvršavaju u aplikaciji treba da imaju niži prioritet od prioriteta ažuriranja stanja koja treba da se dese odmah. U Poglavlju 13, „Ažuriranja stanja visokih performansi“, istražićemo nove API-e koji omogućavaju podešavanje prioriteta ažuriranja stanja u Reactu 18.

Možda se nisu mnogo promenile u Reactu 18, ali dve glavne oblasti koje ćemo razmotriti imaju dalekosežne posledice na način na koji će se React aplikacije implementirati u budućnosti. Postojeći React API-i za ovu verziju uglavnom su nepromenjeni, tako da React zajednica može brzo da usvoji ovu najnoviju veliku nadgradnju verzije bez ikakvih problema.

Rezime

U ovom poglavlju ste detaljno upoznali React. React je biblioteka, sa malim API-em koji se koristi za izradu korisničkog interfejsa. Zatim ste upoznali neke od ključnih koncepata Reacta. Razmotrili smo činjenicu da je React jednostavan, jer nema mnogo neotkrivenih delova. Zatim ste videli deklarativnu prirodu React komponenata i JSX-a. Onda ste saznali da React ozbiljno „shvata“ performanse, pa možete da napišete deklarativni kod koji se može iznova renderovati. Takođe ste učili o ideji ciljeva renderovanja i o tome kako React lako može postati UI alatka po izboru za sve te ciljeve. Na kraju, ukratko sam predstavio što je novo u Reactu 18.

Dosta je uvodnih i konceptualnih „stvari“ za sada. Dok se budemo približavali kraju knjige, ponovo ćemo se osvrnuti na ove ideje. Za sada, hajde da napravimo korak unazad i utvrdimo osnove, počev od JSX-a.

Dodatna literatura

Više informacija pogledajte na sledećim linkovima:

- React: https://reactjs.org/
- React 18: https://reactjs.org/blog/2021/06/08/the-plan-forreact-18.html

2

Renderovanje pomoću JSX-a

U ovom poglavlju ćete upoznati JSX. JSX je XML/HTML sintaksa označavanja koja je ugrađena u JavaScript kod i koristi se za deklarisanje React komponenata. Na najnižem nivou koristićete HTML označavanje da opišete delove svog korisničkog interfejsa. Izrada React aplikacija uključuje organizovanje ovih delova HTML označavanja u komponente. Kada kreirate komponentu, dodajete novi rečnik u JSX, a ne samo osnovno HTML označavanje. Ovde React postaje zanimljiv – na primer, kada imate svoje JSX oznake koje mogu da koriste JavaScript izraze da „ožive" vaše komponente. JSX je jezik koji se koristi za opisivanje korisničkih interfejsa (UI-a) izrađenih pomoću Reacta.

Ovim poglavljem su obuhvaćene sledeće teme:

- prvi JSX sadržaj
- renderovanje HTML-a
- opisivanje strukture korisničkog interfejsa
- kreiranje JSX elemenata
- korišćenje JavaScript izraza
- fragmenti JSX-a

Tehnički zahtevi

Kod za ovo poglavlje možete pronaći u sledećem direktorijumu pratećeg GitHub spremišta: https://github.com/PacktPublishing/React-and- React-Native-4th-Edition/tree/main/Chapter02.

Prvi JSX sadržaj

U ovom odeljku ćemo implementirati obaveznu „Hello, World“ JSX aplikaciju. U ovom trenutku mi samo „uranjamo nožne prste u vodu“; slediće detaljniji primeri. Takođe ćemo razmotriti šta čini ovu sintaksu dobrom za deklarativne strukture korisničkog interfejsa.

Zdravo JSX!

Bez odugovlačenja, evo vaše prve JSX aplikacije:

```
import * as React from "react";
import * as ReactDOM from "react-dom";

const root =
  ReactDOM.createRoot(document.getElementById("root"));

root.render(
  <p>
    Hello, <strong>JSX</strong>
  </p>
);
```

Hajde da pregledamo šta se ovde dešava.

KOD

Funkcija `render()` prihvata JSX kao argument i renderuje ga u DOM čvoru koji je prosleđen funkciji `ReactDOM.createRoot()`.

Stvarni JSX sadržaj u ovom primeru renderuje pasus koji sadrži podebljani tekst. Ovde se ne dešava ništa fensi, tako da smo mogli da umetnemo to označavanje direktno u DOM kao običan znakovni niz. Međutim, cilj je da u ovom primeru pokažemo osnovne korake koji su uključeni u renderovanje JSX-a na stranicu. Sada ćemo da malo razmotrimo deklarativnu strukturu korisničkog interfejsa.

JSX se prevodi u JavaScript iskaze; pregledači ne znaju šta je JSX. Toplo bih preporučio da preuzmete prateći kod za ovu knjigu sa adrese https://github.com/PacktPublishing/Reactand-React-Native-4th-Edition i pokrenite ga dok čitate ovo poglavlje. Sve se automatski prevodi; samo treba da pratite jednostavne korake za instalaciju.

Deklarativne strukture korisničkog interfejsa

Pre nego što nastavimo dalje da razmatramo detaljnije primere koda, hajde da razmislimo o našem primeru „Hello, World". JSX sadržaj je bio kratak i jednostavan. Takođe je bio deklarativan, jer je opisivao šta da treba da se renderuje, a ne kako da se renderuje. Konkretno, ako pogledate JSX, možete videti da će ova komponenta prikazati pasus i neki podebljani tekst u njemu. Ako bi se ovo uradilo imperativno, verovatno bi bili uključeni još neki koraci i verovatno bi morali da se izvrše određenim redosledom.

Smatram da je korisno razmišljati o deklarativnom kao o strukturiranom, a o imperativnom kao o uređenom. Mnogo je lakše dovesti „stvari" u red odgovarajućom strukturom, nego izvoditi korake određenim redosledom.

Primer koji smo upravo implementirali trebalo bi da vam omogući da shvatite šta je deklarativni React. Kako budete čitali ovo poglavlje i knjigu, tako ćete videti da JSX označavanje postaje sve detaljnije. Međutim, ono će uvek opisivati ono što je u korisničkom interfejsu.

Funkcija `render()` ukazuje Reactu da treba da prihvati vaše JSX označavanje i transformiše ga u JavaScript iskaze koji ažuriraju korisnički interfejs na najefikasniji mogući način. Na ovaj način React omogućava da deklarišete strukturu vašeg korisničkog interfejsa, bez potrebe da razmišljate o izvršavanju uređenih koraka za ažuriranje elemenata na ekranu, tj. o pristupu koji često dovodi do grešaka. React uobičajeno podržava standardne HTML oznake koje možete naći na bilo kojoj HTML stranici. Za razliku od statičkog HTML-a, React ima jedinstvene konvencije koje treba poštovati kada koristite HTML oznake.

Renderovanje HTML-a

Na kraju krajeva, posao React komponente je da prikaže HTML u DOM-u pregledača. Zbog toga, JSX ima podršku za uobičajene HTML oznake. U ovom odeljku ćete videti kod koji prikazuje nekoliko dostupnih HTML oznaka. Zatim ćemo razmotriti neke od konvencija koje se obično poštuju u React projektima kada se koriste HTML oznake.

Ugrađene HTML oznake

Kada renderujemo JSX, oznake elemenata ukazuju na React komponente. Pošto bi bilo zamorno kreirati komponente za HTML elemente, React se isporučuje sa HTML komponentama. Možemo da prikažemo bilo koju HTML oznaku u našem JSX-u, a izlaz će biti baš onakav kakav očekujemo.

Sada ćemo da pokušamo da renderujemo neke od ovih oznaka:

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';

const root =
  ReactDOM.createRoot(document.getElementById('root'))

root.render(
  <div>
    <button />
    <code />
    <input />
    <label />
    <p />
    <pre />
    <select />
    <table />
    <ul />
  </div>
);
```

Ne brinite o formatiranju prikazanog izlaza za ovaj primer. Pobrinuli smo se da možemo da renderujemo proizvoljne HTML oznake na očekivani način, bez ikakvih posebnih definicija i uvoza.

Možda ste primetili okolnu oznaku `<div>`, koja grupiše sve ostale oznake kao svoje podređene elemente, zato što je Reactu potreban osnovni element za renderovanje. Kasnije u ovom poglavlju ćete naučiti kako da renderujete susedne elemente bez omotavanja u nadređeni element.

HTML elementi renderovani pomoću JSX-a prate regularnu sintaksu HTML elementa, uz nekoliko suptilnih razlika u prepoznavanju velikih i malih slova i atributa.

Konvencije HTML oznaka

Kada renderujete HTML oznake u JSX označavanju, očekuje se da ćete koristiti mala slova za nazive oznaka. U stvari, pisanje naziva HTML oznaka velikim slovima neće uspeti. Nazivi oznaka prepoznaju velika i mala slova, a elementi koji nisu u HTML-u se pišu velikim slovima. Zbog toga, lako je skenirati označavanje i uočiti ugrađene HTML elemente u odnosu na sve ostalo.

Takođe možete proslediti HTML elementima bilo koje od njihovih standardnih svojstava. Kada im prosledite nešto neočekivano, evidentira se upozorenje o nepoznatom svojstvu. Evo primera koji ilustruje te ideje:

```
import * as React from "react";
import * as ReactDOM from "react-dom";

const root =
  ReactDOM.createRoot(document.getElementById("root"));

root.render(
  <button title="My Button" foo="bar">
    My Button
  </button>
);

root.render(<Button />);
```

Kada pokrenete ovaj primer, neće moći da se prevede, jer React ne „zna“ za element `<Button>`; „zna“ samo za element `<button>`.

Kasnije u knjizi ću razmotriti validaciju svojstava za komponente koje kreirate. Validacijom se izbegava neprimetno loše ponašanje, kao što se vidi u svojstvu `foo` u ovom primeru.

Možete koristiti sve važeće HTML oznake kao JSX oznake, ali ne zaboravite da prepoznaju velika i mala slova i da morate da prosledite odgovarajuće nazive atributa. Osim jednostavnih HTML oznaka, koje imaju samo vrednosti atributa, možete koristiti HTML oznake da biste opisali strukturu sadržaja vaše stranice.

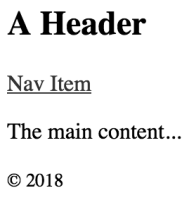
Opisivanje struktura korisničkog interfejsa

JSX može da opiše elemente ekrana na način koji ih povezuje kako bi formirao kompletnu strukturu korisničkog interfejsa. Hajde da pogledamo JSX označavanje koje deklariše složeniju strukturu, a ne samo jedan pasus:

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';

const root =
  ReactDOM.createRoot(document.getElementById('root'));

root.render(
  <section>
    <header>
      <h1>A Header</h1>
    </header>
    <nav>
      <a href="item">Nav Item</a>
    </nav>
    <main>
      <p>The main content...</p>
    </main>
    <footer>
      <small>&copy; 2021</small>
    </footer>
  </section>
);
```

Ovo JSX označavanje opisuje prilično sofisticiranu strukturu korisničkog interfejsa. Ipak, lakše je za čitanje nego imperativni kod, jer je XML, a XML je dobar za koncizno izražavanje hijerarhijske strukture. Želimo da razmišljamo o našem korisničkom interfejsu kada treba da se promeni ne kao o pojedinačnom elementu ili svojstvu, već o korisničkom interfejsu kao celini.

Kako izgleda renderovani sadržaj vidi se na sledećoj slici.



Slika 2.1 Opisivanje struktura HTML oznaka pomoću JSX sintakse

Postoji mnogo semantičkih elemenata u ovom označavanju koji opisuju strukturu korisničkog interfejsa. Na primer, element `<header>` opisuje gornji deo stranice na kojoj se nalazi naslov, a element `<main>` opisuje mesto na kojem se nalazi sadržaj glavne stranice. Ovaj tip složene strukture jasnije predočava programerima dinamičko JSX označavanje. Pre nego što počnete da primenjujete dinamičko JSX označavanje, treba da kreirate neke svoje JSX komponente.

Kreiranje JSX elemenata

Komponente su osnovni gradivni blokovi Reacta. U stvari, one su rečnik JSX označavanja. U ovom odeljku ćete videti kako da enkapsulirate HTML označavanje unutar komponente. Izradićete primere koji ugnežđavaju prilagođene JSX elemente i naučićete kako da kreirate komponente imenskog prostora.

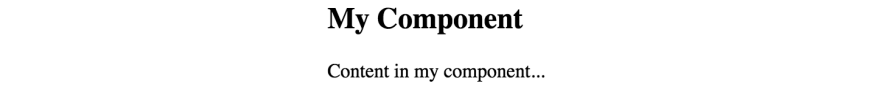
Enkapsulacija HTML-a

Kreiramo nove JSX elemente da bismo mogli da enkapsuliramo veće strukture. To znači da, umesto da unosite složeno označavanje, možete da koristite svoju prilagođenu oznaku. Komponenta React vraća JSX koji se smešta tamo gde je upotrebljena oznaka. Pogledajte sledeći primer:

```
import * as React from "react";
import * as ReactDOM from "react-dom";

class MyComponent extends React.Component {
  render() {
    return (
      <section>
        <h1>My Component</h1>
        <p>Content in my component...</p>
      </section>
```

```
      );
    }
  }

  const root =
    ReactDOM.createRoot(document.getElementById("root"));
  root.render(<MyComponent />);
```

Kako izgleda renderovani izlaz prikazano je na sledećoj slici.



Slika 2.2 Komponenta koja renderuju enkapsulirane HTML oznake

Ovo je prva React komponenta koju smo implementirali, pa ćemo sada izdvojiti vreme da bismo analizirali šta se ovde dešava. Kreirali smo klasu `MyComponent`, koja proširuje klasu `Component` iz Reacta. Na taj način smo kreirali novi JSX element. Kao što možete videti u pozivu funkcije `render()`, renderujete element `<MyComponent>`.

HTML koji ova komponenta enkapsulira vraća metod `render()`. U ovom primeru, kada se JSX prikazuje pomoću `react-doma`, on se zamenjuje elementom `<section>` i svime onim što je u njemu.

Kada React renderuje JSX, svi prilagođeni elementi koje koristite moraju imati odgovarajuću React komponentu u istom opsegu. U prethodnom primeru klasa `MyComponent` je deklarisana u istom opsegu u kojem je deklarisan i poziv funkcije `render()`, tako da je sve funkcionisalo na očekivani način. Obično ćete uvoziti komponente, dodajući ih u odgovarajući opseg. Videćete još mnogo štošta kako budete dalje čitali knjigu.

HTML elementi, kao što je `<div>`, često prihvataju ugnežđene podređene elemente. Sada ćete videti da li možete da uradite isto sa JSX elementima koje kreiramo tako što implementiramo komponente.

Ugnežđeni elementi

Korišćenje JSX označavanja je korisno za opisivanje struktura korisničkog interfejsa koje imaju odnos „podređeni-nadređeni“. Podređeni elementi se kreiraju ugnežđivanjem u drugu komponentu, tj. u nadređeni element. Na primer, oznaka `<li>` je korisna samo kao podređena oznaka oznake `<ul>` ili oznake `<ol>`.Verovatno ćete kreirati slične ugnežđene strukture pomoću vaših React komponenata, pa ćete morate da koristite svojstvo `children`. Sada ćete da vidite kako to funkcioniše. Evo JSX označavanja:

```
import * as React from "react";
import * as ReactDOM from "react-dom";

import MySection from "./MySection";
import MyButton from "./MyButton";

const root =
  ReactDOM.createRoot(document.getElementById("root"));

root.render(
  <MySection>
    <MyButton>My Button Text</MyButton>
  </MySection>
);
```

Uvozite dve vaše React komponente `MySection` i `MyButton`.

Sada, ako pogledate JSX označavanje, primetićete da je `<MyButton>` podređeni element komponente `<MySection>`. Takođe ćete primetiti da komponenta `MyButton` prihvata tekst kao podređeni element, umesto više JSX elemenata.

Hajde da vidimo kako ove komponente funkcionišu, počev od komponente `MySection`:

```
import * as React from "react";

class MySection extends React.Component {
  render() {
    return (
      <section>
        <h2>My Section</h2>
        {this.props.children}
      </section>
    );
  }
}

export default MySection;
```

Ova komponenta prikazuje standardni HTML element `<section>`, naslov, a zatim `{this.props.children}`. Ovo je poslednji deo koji omogućava komponentama da pristupe ugnežđenim elementima ili tekstu i da ih renderuju.

Dve zagrade korišćene u prethodnom primeru se koriste za JavaScript izraze. Više reči o sintaksi JavaScript izraza koja se nalazi u JSX označavanju biće u sledećem odeljku.

Sada pogledajte komponentu `MyButton`:

```
import * as React from "react";

class MyButton extends React.Component {
  render() {
    return <button>{this.props.children}</button>;
  }
}

export default MyButton;
```

Ova komponenta koristi potpuno isti obrazac kao `MySection`; prihvata vrednost `{this.props.children}` i okružuje je oznakama. React se bavi detaljima, a ne mi. U ovom primeru tekst dugmeta je podređeni element `MyButtona`, koji je podređeni element `MySectiona`. Međutim, tekst dugmeta se transparentno prosleđuje `MySectionu`. Drugim rečima, nismo morali da pišemo nikakav kod u `MySectionu` da bismo se uverili da je `MyButton` dobio svoj tekst. Kako izgleda renderovani izlaz prikazano je na sledećoj slici.

Slika 2.3 Element dugmeta renderovan korišćenjem podređenih JSX vrednosti

Možemo dalje da organizujemo naše komponente, tako što ćemo ih postaviti unutar imenskog prostora.

Komponente sa imenskim prostorom

Prilagođeni elementi koje ste do sada kreirali koristili su jednostavne nazive. Imenski prostor obezbeđuje organizacionu jedinicu za vaše komponente, tako da povezane komponente mogu da dele isti prefiks imenskog prostora. Umesto da napišete `<MyComponent>` u vašem JSX označavanju, napisaćete `<MyNamespace.MyComponent>`. To predočava da je `MyComponent` deo `MyNamespacea`.

Obično bi `MyNamespace` takođe bio komponenta. Ideja određivanja imenskog prostora je da komponenta imenskog prostora renderuje svoje podređene komponente, koristeći sintaksu imenskog prostora. Pogledajte primer:

```
import * as React from "react";
import * as ReactDOM from "react-dom";
import MyComponent from "./MyComponent";

const root =
  ReactDOM.createRoot(document.getElementById("root"));

root.render(
  <MyComponent>
    <MyComponent.First />
    <MyComponent.Second />
  </MyComponent>
);
```

Ovo označavanje prikazuje element `<MyComponent>` sa dva podređena elementa. Umesto da pišemo `<First>`, pišemo `<MyComponent.First>`, a umesto da pišemo `<First>`, pišemo `<MyComponent.Second>`. Želimo da eksplicitno pokažemo da `First` i `Second` pripadaju `MyComponentu` unutar označavanja.

Sada pogledajte modul `MyComponent`:

```
import * as React from "react";

class First extends React.Component {
  render() {
    return <p>First...</p>;
  }
}

class Second extends React.Component {
```

```
  render() {
    return <p>Second...</p>;
  }
}

class MyComponent extends React.Component {
  render() {
    return <section>{this.props.children}</section>;
  }
}

MyComponent.First = First;
MyComponent.Second = Second;

export default MyComponent;
export { First, Second };
```

Ovaj modul deklariše `MyComponent` baš kao druge komponente (`First` i `Second`) koje potpadaju pod ovaj imenski prostor. On dodeljuje komponente komponenti imenskog prostora (MyComponent) kao svojstva klase. Postoji nekoliko „stvari" koje možete promeniti u ovom modulu. Na primer, ne morate direktno da izvozite komponente `First` i `Second`, pošto su dostupne pomoću `MyComponenta`. Takođe ne morate da definišete sve u istom modulu; možete da uvezete `First` i `Second` i da ih dodelite kao svojstva klase. Korišćenje imenskih prostora je potpuno opciono; ako ih koristite, trebalo bi da to činite dosledno.

Sada znate kako da kreirate svoje React komponente koje uvode nove JSX oznake u vaše označavanje. Komponente koje smo do sada razmatrali u ovom poglavlju su statične, tj. kada ih jednom renderujemo, nikada se ne ažuriraju. JavaScript izrazi su dinamički delovi JSX-a, koji uzrokuju da React ažurira komponente.

Korišćenje JavaScript izraza

Kao što ste videli u prethodnom odeljku, JSX ima posebnu sintaksu koja omogućava da ugradite JavaScript izraze. Uvek kada React renderuje JSX sadržaj, procenjuju se izrazi u označavanju. To je dinamički aspekt JSX-a i u ovom odeljku ćete naučiti kako da koristite izraze za postavljanje vrednosti svojstava i sadržaja teksta elementa. Takođe ćete naučiti kako da mapirate kolekcije podataka u JSX elemente.

Vrednosti dinamičkih svojstava i teksta

Neke vrednosti HTML svojstava ili teksta su statične, pa se ne menjaju dok se JSX označavanje ponovo renderuje. Ostale vrednosti, vrednosti svojstava ili vrednosti teksta zasnovane su na podacima koji se nalaze negde drugde u aplikaciji. Zapamtite: React je samo sloj prikaza. Sledećim primerom ćemo vam „dočarati“ kako izgleda sintaksa JavaScript izraza u JSX označavanju:

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';

const enabled = false;
const text = 'A Button';
const placeholder = 'input value...';
const size = 50;

const root =
  ReactDOM.createRoot(document.getElementById('root'))

root.render(
  <section>
    <button disabled={!enabled}>{text}</button>
    <input placeholder={placeholder} size={size} />
  </section>
);
```

Sve što je važeći JavaScript izraz, uključujući ugnežđeni JSX, možemo da stavimo između zagrada `{}`. Za svojstva i tekst taj izraz je često naziv promenljive ili svojstvo objekta. Obratite pažnju u ovom primeru da izraz `!enabled` izračunava Bulovu vrednost. Kako izgleda renderovani izlaz prikazano je na sledećoj slici.

Slika 2.4 Dinamička promena vrednosti svojstva dugmeta

Ako pratite kod koji se može preuzeti, što preporučujem da uradite, pokušajte da se „igrate“ ovim vrednostima i videćete kako se renderovani HTML menja.

Primitivne JavaScript vrednosti su jednostavne za upotrebu u JSX sintaksi. Međutim, šta učiniti ako imate objekat ili niz koji treba da transformišete u JSX elemente?

Mapiranje kolekcija u elemente

Ponekad je potrebno da napišete JavaScript izraze koji menjaju strukturu vašeg označavanja. U prethodnom odeljku naučili ste kako da koristite sintaksu JavaScript izraza za dinamičku promenu vrednosti svojstava JSX elemenata. Šta se dešava kada treba da dodate ili uklonite elemente na osnovu JavaScript kolekcija?

U celoj knjizi, kada pominjem JavaScript kolekciju, mislim na obične objekte i nizove. Ili, uopšteno, na bilo šta iterativno.

```
import * as React from "react";
import * as ReactDOM from "react-dom";

const array = ["First", "Second", "Third"];

const object = {
  first: 1,
  second: 2,
  third: 3,
};

const root =
  ReactDOM.createRoot(document.getElementById("root"));

root.render(
  <section>
    <h1>Array</h1>
    <ul>
      {array.map((i) => (
        <li key={i}>{i}</li>
      ))}
    </ul>

    <h1>Object</h1>
    <ul>
```

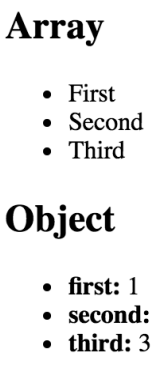
```
        {Object.keys(object).map((i) => (
          <li key={i}>
            <strong>{i}: </strong>
            {object[i]}
          </li>
        ))}
      </ul>
    </section>
  );
```

Prva kolekcija je niz `array` koji je popunjen vrednostima znakovnog niza. Ako se pomerite nadole u kodu do JSX označavanja, možete videti poziv funkcije `array.map()` koji vraća novi niz. Zapravo, funkcija mapiranja vraća JSX element (`<li>`), pa svaka stavka u nizu je sada prikazana u označavanju.

Rezultat procene ovog izraza je niz. Ne brinite – JSX „zna“ kako da renderuje nizove elemenata.

Kolekcija objekata koristi istu tehniku, samo morate pozvati `Object.keys()`, a zatim mapirati ovaj niz. Ono što je dobro u vezi sa mapiranjem kolekcija u JSX elemente na stranici je što možete kontrolisati strukturu React komponenata na osnovu prikupljenih podataka. To znači da se ne morate oslanjati na imperativnu logiku da biste kontrolisali korisnički interfejs.

Kako izgleda renderovani izlaz prikazano je na sledećoj slici.



Slika 2.5 Rezultat mapiranja JavaScript kolekcija u HTML elemente

JavaScript izrazi „oživljavaju“ JSX sadržaj. React procenjuje izraze i ažurira HTML sadržaj na osnovu onoga što je već prikazano i što je promenjeno. Razumevanje načina korišćenja ovih izraza je važno, jer ih svaki React programer svakodnevno najčešće koristi. Sada je vreme da naučite kako da grupišete JSX označavanje bez upotrebe HTML oznaka.

Gradivni fragmenti JSX-a

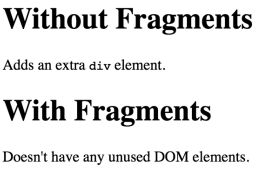
React 16 je uveo koncept JSX fragmenata. Fragmenti služe za grupisanje delova označavanja, bez dodavanja nepotrebne strukture vašoj stranici. Na primer, uobičajeni pristup je da React komponenta vrati sadržaj omotan elementom `<div>`. Ovaj element nema odgovarajuću svrhu i pravi nered u DOM-u.

Sada ćete videti primer. Predstavljam dve verzije komponente. Jedna koristi element omotača, a druga novu funkciju fragmenta:

```
import * as React from "react";
import * as ReactDOM from "react-dom";

import WithoutFragments from "./WithoutFragments";
import WithFragments from "./WithFragments";

const root =
  ReactDOM.createRoot(document.getElementById("root"));

root.render(
  <div>
    <WithoutFragments />
    <WithFragments />
  </div>
);
```

Dva prikazana elementa su `<WithoutFragments>` i `<WithFragments>`.

Kako ti elementi izgledaju kada se renderuju vidi se na sledećoj slici.



Slika 2.6 Fragmenti pomažu u renderovanju manjeg broja HTML oznaka bez ikakvih vizuelnih razlika

Sada ćemo da uporedimo ova dva pristupa.

Korišćenje elemenata omotača

Prvi pristup je omotavanje srodnih elemenata u element `<div>`. Evo kako izgleda izvor:

```
import * as React from "react";

class WithoutFragments extends React.Component {
  render() {
    return (
      <div>
        <h1>Without Fragments</h1>
        <p>
          Adds an extra <code>div</code> element.
        </p>
      </div>
    );
  }
}

export default WithoutFragments;
```

Suština ove komponente su oznake `<h1>` i `<p>`. Ipak, da biste ove oznake vratili iz funkcije `render()`, morate ih omotati pomoću elementa `<div>`. Ispitivanje DOM-a pomoću alatki za razvoj pregledača otkriva da `<div>` ne radi ništa, već samo dodaje još jedan nivo strukture.

```
▼<div>
   <h1>Without Fragments</h1>
  ▼<p>
     "Adds an extra "
     <code>div</code>
     " element."
   </p>
 </div>
```

Slika 2.7 Još jedan nivo strukture u DOM-u

Sada zamislite aplikaciju sa mnogo ovih komponenata – to je mnogo besmislenih elemenata! Pogledajte kako možete da koristite fragmente da biste izbegli nepotrebne oznake.

Korišćenje fragmenata

Sada ćemo pogledati komponentu `WithFragments`, u kojoj smo izbegli korišćenje nepotrebnih oznaka:

```
import * as React from "react";

class WithFragments extends React.Component {
  render() {
    return (
      <>
        <h1>With Fragments</h1>
        <p>Doesn't have any unused DOM elements.</p>
      </>
    );
  }
}

export default WithFragments;
```

Umesto da omotamo sadržaj komponente u element `<div>`, koristimo element `<>`. To je poseban tip elementa koji ukazuje da samo njegovi podređeni elementi treba da budu renderovani. Možete videti razliku u odnosu na komponentu `WithoutFragments` ako pregledate DOM.

```
<h1>With Fragments</h1>
<p>Doesn't have any unused DOM elements.</p>
```

Slika 2.8 Manje HTML-a u fragmentu

Od pojave fragmenata u JSX označavanju imamo manje prikazanog HTML-a na stranici, jer ne moramo da koristimo oznake `<div>` samo za grupisanje elemenata, pa kada komponenta renderuje fragment, React „zna“ da treba da renderuje podređeni element fragmenta gde god se komponenta koristi.

Dakle, fragmenti omogućavaju React komponentama da renderuju samo važne elemente; na renderovanoj stranici se više neće pojavljivati elementi koji nemaju svrhu.

Rezime

U ovom poglavlju ste naučili osnove JSX-a, uključujući njegovu deklarativnu strukturu, koja omogućava lakše održavanje koda. Zatim ste napisali kod za prikazivanje osnovnog HTML-a i upoznali ste kako se opisuju složene strukture, koristeći JSX; svaka React aplikacija ima najmanje jednu strukturu.

Takođe ste učili o proširenju rečnika JSX označavanja implementacije React komponenata da biste dizajnirali svoj korisnički interfejs kao niz manjih delova i „lepili" ih da biste formirali celinu. Zatim ste saznali kako možete da unesete dinamički sadržaj u svojstva JSX elementa i kako da mapirate JavaScript kolekcije u JSX elemente, eliminišući potrebu za imperativnom logikom za kontrolu prikaza korisničkog interfejsa. Na kraju ste naučili kako da renderujete fragmente JSX sadržaja – koristi se nova React 16 funkcionalnost, pri čemu se sprečava upotreba nepotrebnih HTML elemenata.

Sada, kada znate kako se renderuju korisnički interfejsi (UI-ji) ugrađivanjem deklarativnog XML-a u JavaScript module, pređimo na sledeće poglavlje, u kojem ćemo dublje „zaroniti" u svojstva i stanje komponenata.

Dodatna literatura

Više informacija pogledajte na sledećim linkovima:

- Uvod u JSX: https://reactjs.org/docs/introducing-jsx.html
- Fragmenti: https://reactjs.org/docs/fragments.html