Lab. vježba

11

NAPREDNE WEB TEHNOLOGIJE

Upute za laboratorijske vježbe (izradio: Toni Beović)

React

# Zadatak

O

tvorite <https://reactjs.org/docs/add-react-to-a-new-app.html> i pratite upute kako kreirati React projekt korak po korak.

Prvo je potrebno instalirati create-react-app paket:

npm install -g create-react-app

Nakon toga, pokrenite sljedeću naredbu i zamijenite my-app s proizvoljnim imenom vašeg projekta:

create-react-app my-app

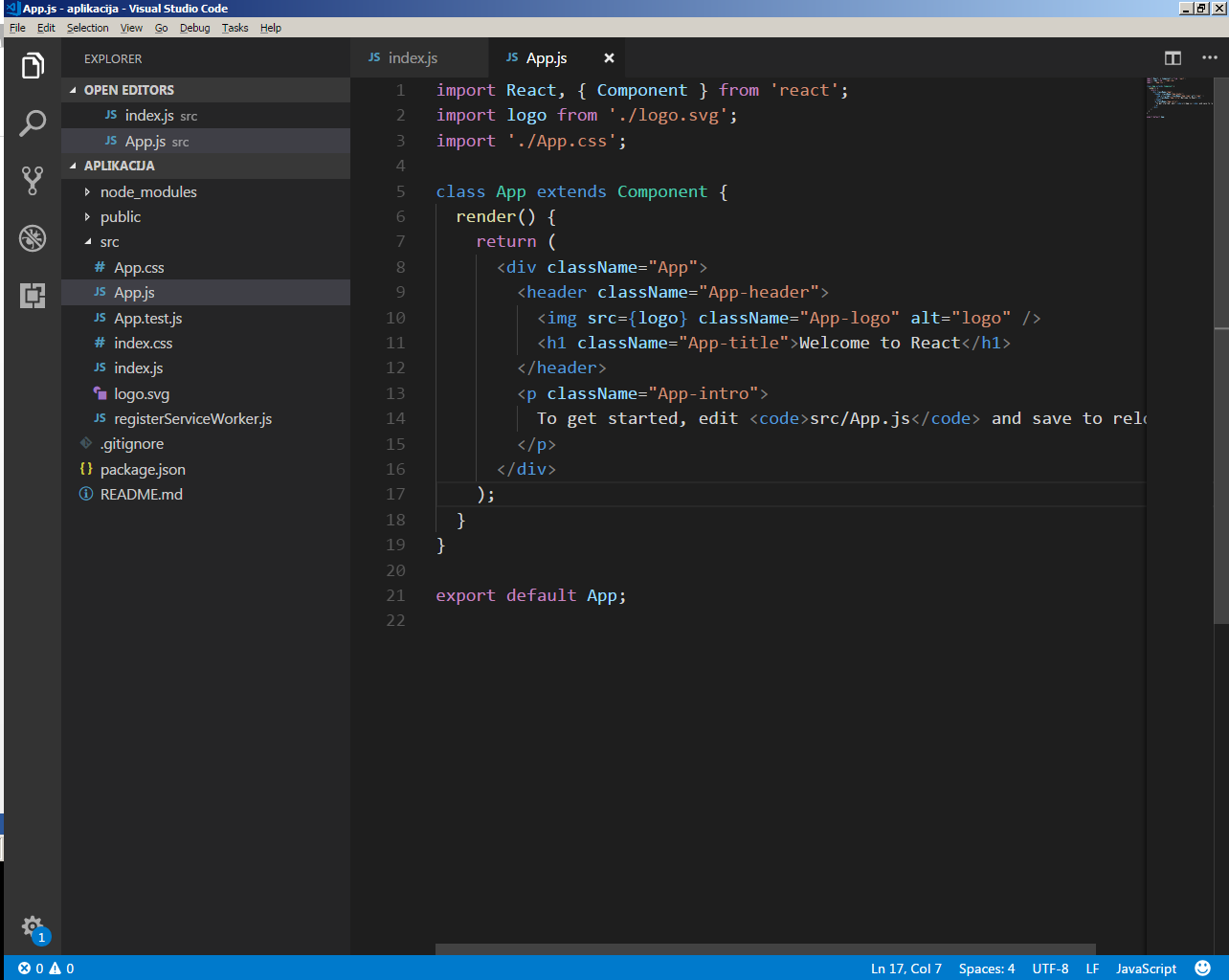
Zatim uđite u direktorij

cd <ime vašeg direktorija>

I pokrenite aplikaciju:

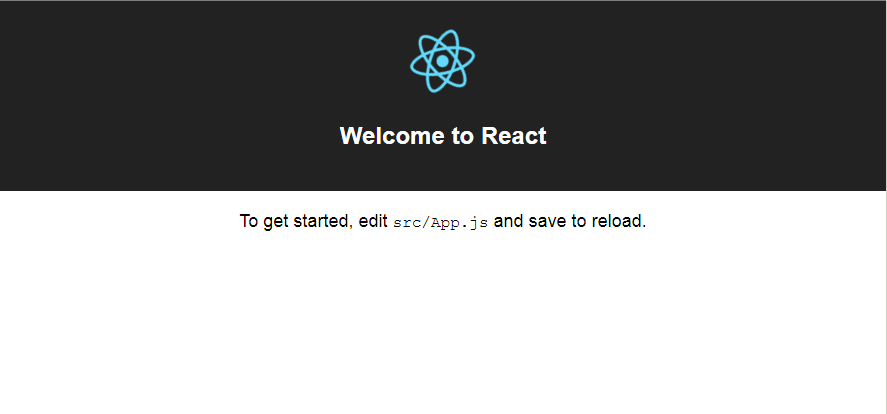
npm start

Pokrenite Visual Studio Code i otvorite vaš direktorij



Možete otvoriti integrirani terminal unutar Visual Studio Code-a klikom na View>Integrated terminal

Ako ste sve dobro obavili, pokrenut ćete svoju prvu React aplikaciju:



# Komponente

Komponente su temeljni dio Reacta. Omogućavaju odvajanje korisničkog sučelja (UI) u neovisne, ponovno iskoristive dijelove. Komponente grade stranicu pomoću običnih elemenata kojima postavljaju funkcionalnost.

Nasljeđivanjem Component klase Reacta, ES6 klasa se definira kao komponenta. Komponente možemo zamisliti poput običnih JavaScript funkcija. Primaju proizvoljne ulazne podatke zvane „props“ (od riječi „properties“) i vraćaju React elemente koji opisuju ono što se treba ispisati na ekranu. Komponente mogu referencirati ostale komponente u svom izlazu (output). Na taj način možemo jednu te istu komponentu koristiti na više mjesta. Obično React aplikacije imaju jedinstvenu App komponentu kao temeljnu (root) komponentu, te se sve ostale nadovezuju na nju. Komponente se obično razbijaju na vrlo nisku razine, poput polja formi ili dugmad. Osim već spomenutih ES6 Class komponenti, postoje i funkcionalne komponente, koje su u suštini samo Javascript funkcije.

# Renderiranje

React koristi JSX, sintaktički dodatak Javascriptu za opisivanje izgleda. Obično koristimo JSX kako bismo definirali povratnu vrijednost komponente, odnosno ono što će ona prikazati. JSX kao izlaz generira React elemente. Za razliku od Angulara u kojem se odvaja predložak od logike aplikacije, u Reactu to nije slučaj. JSX izgleda kao kombinacija običnog HTML-a i dodatnih elemenata, koje koristimo kako bismo pozvali vlastite komponente. Unutar JSX koda moguće je izvršavati bilo kakav JS izraz stavljanjem vitičastih zagrada oko nj. React automatski ponovno iscrtava samo one elemente koji su se promijenili na stranici. Da bi mogli određeni sadržaj renderirati, potrebno ga je staviti unutar jednog JSX elementa i pozvati render metodu.

# Stanja i životni ciklus

Svaka React komponenta prolazi kroz nekoliko metoda koje definiraju njezin životni ciklus. Ove metode je moguće preopteretiti i izvršiti proizvoljnu funkcionalnost u određenom trenutku životnog ciklusa. Sam ciklus sastoji se od tri jednostavna dijela: uključivanja (mounting), ažuriranja (updating) i isključivanja (unmounting). Metode koje su vezane uz postupak uključivanja pozivaju se kad se instanca komponente stvori i proslijedi DOM-u. Ažuriranje može biti izazvano mijenjanjem stanja ili propsa. Metode isključivanja pozivaju se kad se komponenta briše iz DOM-a. Tok podataka u Reactu je jednosmjeran i temelji se na predavanju propsa, odnosno svojstava komponenti više razine na komponente niže razine.

State je poseban React objekt koji uzrokuje ponovno iscrtavanje sadržaja kad se promijeni. Svaka komponenta ima vlastiti state objekt, odnosno svaka komponenta zasebno kontrolira vlastito stanje. U state objektu držimo informacije koje želimo dinamički mijenjati, odnosno informacije koje utječu na izlaz komponente. Promijene stanja su asinkrone. Neke od bitnijih metoda su:

* **setState()** – koristi se za modificiranje state objekta komponente i govori Reactu da komponenta i njezina djeca moraju biti ponovno iscrtane s ažuriranim stanjem.
* **render()** – koristi se za iscrtavanje elemenata, specificira povratnu vrijednost komponente
* **constructor()** – konstruktor ES6 klase, poziva se kada se komponenta stvori.
* **componentWillMount()** – poziva se prije nego što se komponenta učita. Korisno za postavljanje inicijalnog stanja.
* **componentDidUpdate()** – poziva se u trenutku kad se komponenta ažurirala
* **componentWillReceiveProps()** – poziva se prije nego što komponenta primi svojstva

# Zadatak

Implementirati jednostavnu Todo aplikaciju koja se sastoji od:

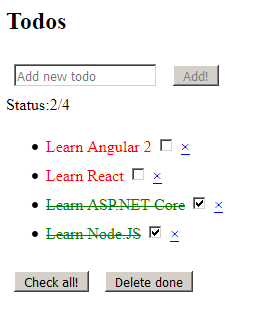
• Popis stavki „todo“ stavki

• Mogućnost dodavanja „todo“ stavki

• Mogućnost brisanja „todo“ stavki

• Označavanje pojedine stavke kao obavljene

Aplikacija treba izgledati kao na slici:



1. Kreirati „**models**“ direktorij i unutar njega kreirati **todo.js** datoteku u kojoj je

eksportirana klasa **Todo** s dva svojstva:

* 1. text
  2. done

1. Modificirati **App.js**:
   1. Importirati klasu **Todo**
   2. Kreirati niz **todos** te ga u **componentWillMount** postaviti u stanje pozivom **setState** metode. Niz treba sadržavati nekoliko **Todo** stavki od kojih su neke popunjene, a neke ne
   3. Definirati listu koju je potrebno popuniti stavkama s **Todo** tekstom
   4. Dodati labelu koja će ispisati koliko je Todo riješeno i koliko ih je ukupno (Status: 2/4)
   5. Dodati klase kojim će se razlikovati riješene od neriješenih Todo modificiranjem **App.css** datoteke
   6. Za svaki Todo dodati checkbox kojim se isti može označiti kao riješeni, odnosno neriješeni
   7. Za svaki Todo dodati „x“ gumb kojim se omogućuje brisanje istog
   8. Iznad popisa stavki dodati input text element i button Add. Novi Todo se dodaje

ako je na neprazan input pritisnuta tipka enter ili ako je kliknut Add button.

Dodavanjem nove stavke izbriše se trenutni tekst u input elementu.

* 1. Dodati button «Check All», s kojim se sve stavke mogu označiti kao obavljene
  2. Dodati button «Delete done» s kojim se brišu sve odrađene stavke