**React** je jezik za frontend, single page je, radi se s komponentama

Dolazi uz biblioteku JSX koja izgleda maltene identično kao HTML, a uloga toga jeste da se react.js (koji je JavaScript lib) mogu slati elementi koji izgledaju isto kao HTML (ali nisu zapravo HTML).

- po defaultu radi na portu 3000

- aplikacija se pokreće sa npm start komandom u CMD

- nema potrebe za refreshovanjem stranice dok radimo; sama se azurira cim spasimo promjene

- sa function se prave komponente

- importi uvijek moraju imati relativni put sa ./ime\_fajla

- importi uvijek pretpostavljaju da je ekstenzija .js

funtcion Nesto()

{

return (//kod);

}

- u Dom.render() možemo koristiti komponentu nakon što je ovako napravimo

- you can style your components, samo sto nije ista sintaksa; u CSS-u je nesto-nesto, ovdje je nestoNesto (camel case notacija)

JSX

- ne mogu se renderovati dva susjedna JSX elementa, moraju se enkapsulirati u nešto, kao u div npr i onda poslati render funkciji

REACT I SLIKE

React ne može da radi s lokalnim slikama samo tako. Pogotovo ako se koristi WebPack (which I do). Stvar je u tome što kad bi se build uradio, relativne putanje do slike više ne bi funkcionisale ako se samo stavi <img src=“./kjdcnkjd“/>

Vanjske slike rade normala btw, problem je samo sa slikama koje su lokalno spašene na računar.

Fix: <https://github.com/webpack-contrib/file-loader>

KOMPONENTE

Što se komponenti tiče, postoje 2 tipa: functional i class-based.

Functional components su komponente koje su u suštini funkcija koja kao rezultat vraća UI komponentu koju smo napravili. Lagane su za razumjeti jer ono, funkcije.

class-based:

class App extends React.Component

{

render(){

return();

}

}

Svaka class-based komponenta MORA imati render metodu. Ovakve komponente nam služe da napravimo display logiku - to se radi u render-u, prije return metode. Pored render metode, možemo imati koliko god hoćemo metoda (kao kod obične klase). Pozivaju se unutar render metode.

Kod class-based komponenti moramo pisati this.props ako želimo da koristimo props.

PROPS i STATE

Props are immutable!

States su samo za class-based komponente. Prvo se doda konstruktor:

constructor()  
{  
 super();

this.state ={

property\_name: „“  
}

}

super metoda se UVIJEK koristi sa konstruktorom. Ona grabi neke goodies iz klase. U this.state se inicijaliziraju svi properties klase na neke početne vrijednosti.

Ovako se pristupa:

<ChildComponent neki\_prop={this.state.property\_name}

State je različit od props ato što se state smije mijenjati. Za to nam služi metoda setState i kad je god koristimo potrebno je da uradimo binding metode koja koristi setState i klase kojoj ta metoda pripada.

BINDING

Ako hoćemo da koristimo custom metode unutar neke klase, a da ta metoda mijenja state, MORAMO uraditi binding.

State NE MOŽEMO mijenjati direktno! Ne možemo reći this.state.proprtyName = nesto

Nego prvo radimo binding:

unutar klase:   
this.nazivMetode = this.nazivMetode.bind(this);  
  
Onda unutar nazivMetode radimo nešto sasu sa setState i onda negdje iz render() metode pozivamo nazivMetode da uradi posao za nas i promijeni state.

setState metodi uvijek šaljemo lambda fju (može i imenovana ali praksa je da bude lamda) koja u sebi sadrži parametar koji se zove kako god mi hoćemo ali u koji metoda super() sama od sebe ugura previous state da ga možemo koristiti. Ovo je super useful jer je često bitno da znamo koje stanje je prethodilo ovom u kojem se klasa trenutno nalazi.

U lambda fji radimo šta nam je ćeif.