***Udemy-REACT-1***

Dobar izbor kazu ljudi

https://react.new to ukucas u browser i preko browsera mozes da kucas react kod (onlice visual code).

REACT KOMPONENTA je jedan delic user interfejsa.

Treba ti Prettier ekstenzija za lakse citanje koda.

import {useState} from "react";

export default function App() {

const [advice, setAdvice] = useState("");

async function getAdvice(){

const res = await fetch('https://api.adviceslip.com/advice');

const data = await res.json();

setAdvice(data.slip.advice);

}

return (

<div>

<h1>{advice}</h1>

<button onClick={getAdvice}> Klikni me!</button>

</div>

);

}

U button mozes da ubacis onClick={getAdvice} koja poziva getAdvice metodu svaki put kada se klikne da dugme.

**STATE KONCEPT** je najosnovniji koncept u React-u. Svaki put kada se nesto menja u user interfejsu mi treba da promenimo state. Svaka promena u user interfejsu updejtuje nesto sto se zove stat.

**STATE** je objekat koji cuva informacije o komponenti koja se moze menjati tokom vremena.

Znaci state je objekat koji ima neki value i neku seter metoda za updejtovanje tog value. Kada se poziva seter metoda kao parametar se prenosi vrednost koja ce biti upisana u value. I tako se pri svakom pozivu upisuje neka vrednost.

**useState** je funkcija koja se importuje iz react biblioteke, ona vraca neki Array koji se destruktuje na [advice, setAddvice]

advice- value odnosno vrednost stat-a

setAdvice-setter funkcija odnosno funkcija za updejtovanje stat-a.

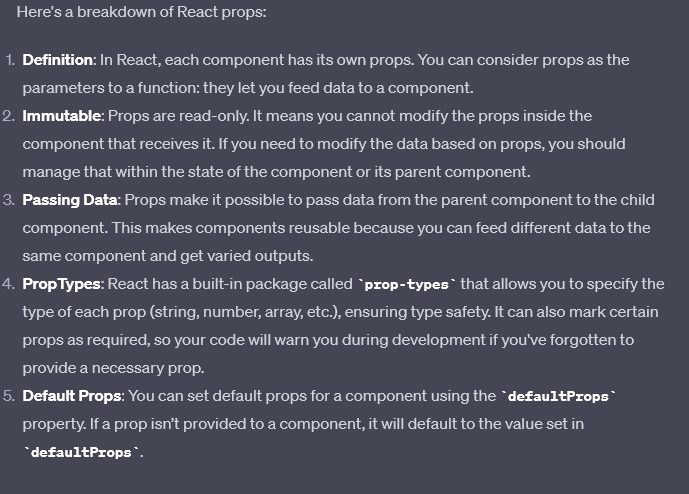
const [count, setCount] = useState(0);

setCount((c => c + 1));

Znaci mozes da ga napravis i kao brojac, gde svaki put kada kliknes se count inkrementira za 1.

useEffect ima dva parametra, prvi je funkcija koju zelimo da se izvrsi odmah na pocetku, a drugi argument je dependencyArray, za pocetak je prazni string. To je neki hook.

Kroz REACT cilj je da korisnicki interfejs podelis na **KOMPONENTE**. A svaka komponenta je neka funkcija. Komponente su building blocks svake react aplikacije. Omogucavaju da enkapsuliras logiku, ponasalje, stilove unutar delica koji se mogu eksportovati i koristiti veci broj puta po potrebi. Ima klasne komponente i funkcijske komponente.

REACT PROPS, props (property) su nesto kao parametri za funckiju. Koriste se za prenos vrednosti sa roditeljske na deciju komponentu.

<Message count={count}/>

Komponenti Message kao parametar preko props-a prenosis vrednost count.

function Message(props){

return(

<p>

Kliknuo si <strong>{props.count}</strong> puta

</p>

);

}

Komponenta Message preko props.count moze da pristupi toj vrednosti. Vrednost je read-only, dete ne moze da je menja.

Linkovi neki:

* <https://vitejs.dev/guide/>
* <https://react.dev/>
* <https://create-react-app.dev/>

***SECTION 3:***

Zasto JS frejmvorci uopste postoje kao sto je React? Zasto se za sve ne koristi vanila js?

Znaci u pocetku klijent bi slao zahtev serveru za nekom web stranicom, server bi izrenderovao i kao odgovor slao klijentu html, css, js fajlove i browser klijenta bi samo interpetirao kod i ucitao stranicu. A sa pojavom single-page aplikacija, renderovanje se vrsi na klijentskoj strani sada podaci stizu preko API-a. To omogucava da se kompletna web aplikacija izvrsava samo na jednom html stranici.

Ima problema kod razvoja single page aplikacija sa vanila javascriptom. Front end deo treba da prihvati neke podatke i da ih prikaze korisniku na ekranu. Interfejs treba da prikazuje uvek trenutno stanje sa podacima. Potrebno je da korisnik uvek ima najnoviju verziju podataka.

-> PIECE OF DATA == PIECE OF STATE

Bez odredjenih frejmworka je skoro pa nemoguce da se ogromne kolicine podataka sinhronizuju sa super-kompleksnim korisnickim interfejsom. Jednsotavno vanila js bi bio davao jako nepregledan i tezak kod za razumevanje.

Zasto front-end frejmworci uopste postoje?

Postoje kako bi se olaksalo sinhronizacija korisnickih in terfejsa sa podacima sa kojima web aplikacija radi. Fremworci olaksavaju posao programerima. Razliciti frejmworci ima razlicite pristupe resenju ali svi imaju zajednicki cilj a to je da podaci i UI (User interface) budu sinhronizovani. Oni takodje pruzaju pravilan nacin za pisanje i strukturu koda kako bi kod bio sto laksi za razumevanje. Moderni web dev se vrti oko js frejmworka.

Znaci kod React sve je odradjeno kroz js dok kod vanila js se idalje koriste html elementi.

**RAZLIKA JE** kad se kod vanila js vrsi promena vrednosti neke promenljiva ona nece da se odmah odrazi na korisnicki interfejs vec je neophodno da programer manuelno onese novu vrednost u interfejs, dok se sa react-om koriste states gde svaka promena nad statom automatski vrsi updejt korisnickog interfejsa. Znaci velika prednost REACT u odnosu na vanila js jesu **STATES**. Znaci veoma je bitno da vrednosti koje se prikazuju na UI budu sinhronizovane sa DATA. React smanjuje kolicinu koda u web aplikacijama.

**STA JE REACT?**

React je javascript biblioteka za izgradnju user interfejsa. Bazira se na komponentama i state-u. Kreiran od strane fejsbuka i veoma je popularan. Reakt je skup komponenti. Komponente su gradivni blokovi UI u react-u. Komponenta moze biti neko dugme, heder, div sa drugim komponentama i tako dalje. React uzima komponente i kreira web app na osnovu njih.

**JSX** je deklarativna sintaksa. Koristi se za opis kako komponente izgledaju na osnovu trenutnog stanja. React je apstrakcija DOM-a. Nikad se ne radi sa DOM direktno kao u vanilla js. JSX je sintaksa koja kombonuje html, css, js i omogucava struktuiranje komponenti.

Glavni cilj React-a je da UI bude u skladu sa DATA. Na osnovu trenutnog stanja REACT renderuje UI, onda se stanje moze promeniti nakon klika na dugme, svaki put kad se stanje promeni react automatski re-renderuje UI kako bi se reflektovale promene na korisnicki interfejs. Znaci nakon svake promene STATE, react re-renderuje UI.

**REACT REACTS TO STATE CHANGES BY RE-RENDERING THE UI.**

Zbog toga se React zove React. Da li je React biblioteka ili framework. On je ustvari samo biblioteka. Jer biblioteka je alat a framework je skup alata. A React je samo jedan alat za izgradnju SPA aplikacija. Da bi se izgradila aplikacija potrebni su jos neki dodatni alati kao sto je Next.js.

React je popularan i ostace popularan jer veliki broj kompanija, cak i vecih, su prihvatile react odavno. Sve vise manje kompanije koriste React. Zato nastaje velika potreba za React poslovima i to su dobro placeni poslovi. Sada postoji i React community. Uvek ce biti tutoriala, pitanja i pomoci oko react dev-a.

Takodje veliki broj 3-rd party alata mozes da inkludujes u svoju react aplikaciju.

React je kompletno transformisao moderni web dev. React je dobar za renderovanje komponenti na UI na osnovu trenutnog stanja.Takodje je jako dobar i odrzava sinhronizaciju DATA sa UI jer reaktuje svaki put kad se izvrsi promena nad state rerenderuje se UI. UI prati svaku promenu state-a.

Znaci treba imas Visual Studio Code i Node.js novu verziju.

**ESLint** ekstenzija obavezno (MNOGO JAKO, AUTOMATSKI TI FORMATIRA KOD)

**Prettier** jer lepo formatira kod

**Material Icon Theme** isto ekstenzija koju mora skines

Settings -> default format -> Editor: Default Formatter -> **Prettier (JAKO BITNO)**

Settings -> format on save -> check

Settings -> eslint run -> run -> onSave //eslint proverava kod svaki put kad ga zapamtimo

Treba instaliras i najnoviju verziju node // node –v da proveris koja je verzija

Treba neki **snippet** da napravis, da prekopiras nesto

***PURE REACT***

**HTML TEMPLATE** u VsCode: Ukucas ! + Enter

root Div ima className root.

Ukljucis dve biblioteke, prva je kao neki interfejs a druga je rendering layer koja renderuje react komponente u DOM.

<script src="https://unpkg.com/react@18/umd/react.development.js"></script>

<script src="https://unpkg.com/react-dom@18/umd/react dom.development.js"></script>

Iz ovih biblioteka se uzima React objekat i metode koje se pozivaju nad njim.

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

root.render(React.createElement(App)); //Iscrtava komponentu App unutar root-a, ovako se ne radi nego se koristi JSX sintaksa, nego bajca samo da pokaze da moze.

file->preferences->settings-> Type **editor.copyWithSyntaxHighlighting->** untick the box labelled **Controls whether syntax //Kad kopiras iz VS da nemas crnu ppozadinu.**

**useEffect je hook koji se aktivira uvek kada dodje do necega do cega treba da dodje. U ovom primeru useEffect se aktivira svaki put kada prodje 1000ms i tada poziva setTime koja menja state time.**

**const [time, setTime] = React.useState(new Date().toLocaleTimeString());**

**React.useEffect(function () {**

**setInterval(function () {**

**setTime(new Date().toLocaleTimeString());**

**}, 1000);**

**}, []);**

**->Escape Hatches je dobro poglavlje u react.dev dokumentaciji.**

**->DOBAR DEVELOPER UVEK ZNA GDE DA NADJE INFORMACIJE KOJE SU MU POTREBNE a officialna dokumentacija je cesto najbolji izbor.**

**Za kreiranje React App imas dve opcije:**

**1. ) CREATE-REACT-APP (kompetan “starter kit” za react app, sve je vec configurisano (ESLint, Prettier, Jest, etc...) problem je sto je napravljeno davno i koristi spore i outdated tehnologije, narocito webpack budles. Trenutno preporuka je da se ovo vise ne koristi za projekte jer nema vise podrske za to, jedino za tutoriale da se koristi. NE KORISTI ZA PRAVE APLIKACIJE.**

**2. ) VITE je perfektan izbot za izgradnju React app. Moderan build tool koji sadrzi template za podesavanje React aplikacije. Problem je sto moras manuelno da podesis Prettier, EsLint, Jester i ostale zebancije. VITE je ekstremno brz u refresovanjus stranice kada se desi neka promena (hot module replacement). Bundiling je isto jako brzo. Stranica se instantno refresuje svaki put kada sejvujemo kod.**

**React ima frejmworke kao sto su Next.js i Remix. Da bi naucio Next.js treba prvo React da znas.**

**ZA KREIRANJE PREKO PRVOG NACINA KUCAS ->**

**npx create-react-app@5 imeAplikacije //@5 oznacava verziju 5 react-a, da bi radilo treba i ovo iskucas-**npm install -g npm@latest

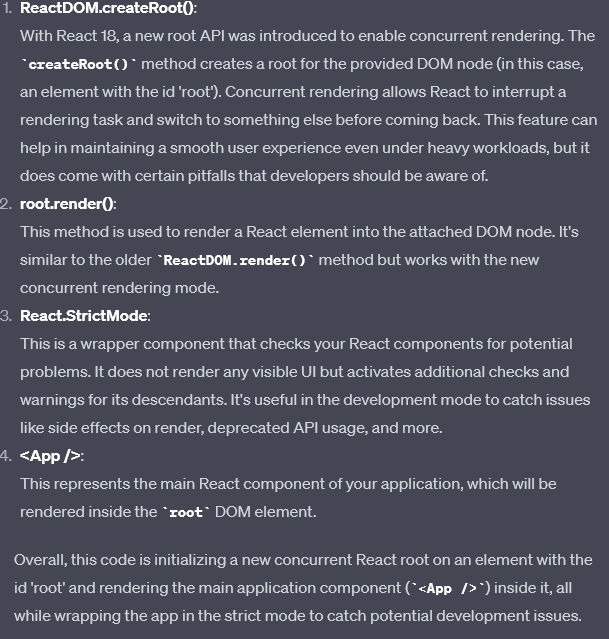
npm cache clean --force

npm rm -rf node\_modules && rm package-lock.json

npm install

npm install create-react-app

npx create-react-app myapp

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

**root.render(**

**<React.StrictMode>**

**<App />**

**</React.StrictMode>**

**);**

**//KREIRA I RENDERUJE ROOT**

U reactu nije bitno gde se sta nalazi, koji fajlovi u kojim folderima... sve je proizvoljno.

Da bi pokrenuo React app treba da ukucas: npm run start (pokrece skriptu “start” koja se nalazi u package.json)

***SECTION 4 js obnavljanje***

React je javascript biblioteka. U section4 ce da obnove ono najbitnije za javascript kao sto su:

Destructing, spread operator, template literals, ternaries, promises, Async/Await, Map,Filter,Reduce,Sort

Sta su immutable arrays? Map, Filter,Reduce,Sort jel to?

* Treba skines ekstenziju **Quokka** (omogucava da se skripta izvrsava direktno iz Vs Coda) da pokrenes treba Manage -> Command pallete -> Quokka start on Curent file

--Ovde obnavljaju JavaScript

ObjectDestructuring: const {ime, prezime,pages,genres} = book;

-----------------------------------------------------------------------------------------------------------------------------------------

***SECTION 5***

Kako se kreiraju i ponovno koriste komponente koriscenjem JSX sintakse.

Rendering lists

Conditional rendering

Pravi se pizza-menu. Prvi fajl uvek mora da ima naziv index.js. Sve iz foldera src izbrisi i napravi novi fajl.

//React v18

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

root.render(

  <React.StrictMode>

    <App />

  </React.StrictMode>

);

Strict mode ce renderuje dva puta kako bi pronasao potencijalne greske. Po konvenciji bi trebalo da se pise ali nije obavezan.

AKO NECE DA SE POKRENE SA //npm run start UKUCAJ //npm install react-scripts –save

**DEBUGAVANJE: STA RADITI KADA NESTO NECE?**

1. Proveri da li je aplikacija pokrenuta

2. Prekini aplikaciju sa ctrl + c i pokreni ponovo

3. Relouduj u browseru jer hot reload nekad nece da radi

4. Terminal uvek da bude otvoren u Vs Code i Console u browseru (dev tools), greske se uvek tu prikazu

5. Kad naidjes na gresku, ako ti nije jasna odmah googluj

6. Uvek radi sa ESLint

7. Uvek proveri zute linije u kodu

8. U output mozes da vidis da li je negde problem sa necim

**KOMPONENTE:**

Komponente su najosnovni koncept u React-u. React aplikacije su kompletno napravljene iz komponenti. U bilo kojoj React app nema nista sto nije komponenta ili nije deo neke komponente. Zato su komponente “building blocks” za korisnicki interfejs u React-u. Reacti uzima komponente i kreira UI interfejs. React renderuje pogled na svaku komponentu i svi pogledi zajedno cine korisnicki interfejs. Komponenta je deo korisnickog interfejsa. Svaka komponenta ima svoj: Data, JS Logic i Apperance. Svaka komponenta ima nacin kako radi i kako izgleda. Kompleksni UI se dobija kreiranjem veceg broja komponenti i onda kombinovanjem kao kockice u slagalici.

KOMPONENTE SE NAJCESCE STAVLJAJU UNUTAR NEKIH DRUGIH KOMPONENTI. (We nest components inside each other). Ponovno koriscenje komponenti i njihova ugradnja unutar drugih komponenti su kljucni aspekti react-a. Svaki put kad je potrebno da se nesto vise puta iscrta na UI, napravis jednu komponentu i koristis je koliko puta hoces. A podatke prenosis preko props. Kljucan skill je kako da dizajn jedne web strane razbijes na komponente. Pravis “COMPONENT TREE”.

Komponente su funkcije koje pocinju velikim slovom i uvek moraju da vrati neki MARKUP u JSX formatu. Ili moze da vrati null.

SVAKA KOMPONENTA MOZE DA VRATI SAMO JEDAN ELEMENT. NIKAKO DVA ILI VISE. Nesting nije da pises funkciju unutar funkcije. NE PISI FUNKCIJU UNTAR FUNKCIJE. Nesting podrazumeva da se komponenta ukljucuje unutar komponente.

function App() {

  return (

    <div>

      <h1>Hello friend</h1>

      <Pizza />

    </div>);}

IRL se javascript kod deli u module i onda se modulu ukljucuju jedan u drugi ali u ovom primeru se sve pise u jedan fajl index.js.

 <img src="pizzas/spinaci.jpg" alt="Pizza" />

Slika obavezno ima src i alt property.

settings -> diff decorations -> none //Kako bi se izgubile linije zbog git repository sa strane

**What is JSX?**

Svaka komponenta sadrzi Data, Logic, Apperance. JSX je deklerativna sintaksa koja se koristi za opis kako komponenta izgleda i kako radi na osnovu njenih podataka i logike. JSX se koristi ja prikaz komponente. Svaka komponenta vraca jedan blok JSX koji react renderuje i prikazuje UI. JSX je extenzija javascript-a, koja omogucava da se iskombinuje html, css, i js u jedan blok koda. Znaci omogucava da pises pregledan spageti kod.

**BABEL** vrsi konverziju JSX u javascript. Svaki JSX element se konvertuje u React.createElement poziv f-je. Ova konverzija je neophodna jer browseri ne razumeju JSX, oni samo razumeju HTML. Znaci React prevodi JSX u React.createElement f-je i onda se pozivom tih f-ja kreiraju HTML elementi koji se mogu interpetirati u browseru. React se moze koristiti bez JSX ali je to mnogo komplikovaniji nacin i kod se mnogo teze cita i razume. Svi koriste JSX u praksi. JSX je deklerativna sintaksa.

**JSX IS DECLERATIVE** (ima imperativan i deklarativan pristup)

**Imperative** znaci da se manuelno selektuju DOM elementi, vrsi se DOM traversing, i dodaju se EventHandlers na elemente. Svaki put kad se de si neki dogadjaj, npr klik na dugme, mi govorimo browseri korak po korak koje sve operacije treba da se izvrse da bi se dobio zeljeni UI. Na imperativan nacin mi govorimo browseru kako i sta da uradi.

**Deklaritivan** pristup je kako UI treba da izgleda svaki put na osnovu trenutnih podataka.

DATA = props and state. Mi koristimo JSX da opisemo UI na osnovu props i state. Sve se to odvija bez ikakve DOM manipulacije. Nema querySelectori, eventListeneri, classList, textContent... React je velika apstrakcija od DOM-a. Omogucava programerima da nikad ne interaguju sa DOM-om direktno. Razlika izmedju imperativnog i deklarativnog pristupa je sto u deklarativnom pristupu mi koristimo JSX da kazemo React-u sta zelimo da vidimo na ekranu, ali ne i kako to da ostvari korak po korak. React to sam moze da zakljuci. I ovo ima puno prednosti.

Za deklaraciju komponenti mozes da koristis Function expression i arrow funkcije ali se obicno prave sa kljucnom reci function. Lepota JSX je sto mozes da pises komponente kao HTML komponente.

Kompleksni UI se postize kombinovanjem manjih komponenti unutar vecih komponenti.

{new Date().toLocaleTimeString()}

Vraca trenutno vreme.

Komponente su JS f-je i moguce je pisati bilo koji JS kod unutar njih. Cim se komponenta inicijalizuje taj kod se izvrsava.

Zbog strikt mode se dva puta poziva alert? U strikt modu komponente se renderuju dva puta.

JSX kombinuje css, html i js u jedan jednistven kod. REACT HAS SEPARATION OF CONCERNS.

Postoji puno nacina za stilizovanje komponenti. (inline, external. SAS, css modules, Tailwind css...)

Ima <main> HTML element. Stilove moras da importujes.

import "./index.css";

**PROPS**

Props se koriste za prenos DATA izmedju komponenti. Props je komunikacioni kanal izmedju parent i child komponente. Props su najosnovnija stvar u React-u. Props su argumenti koji se prenose.

function Komponenta(props){ props.ime...)

Znaci skoro svaka komponena ima neki props. To je kolekcija parametara. Objekat. Key:Value. JS Object..

{javaScriptMode}, znaci da pises js kod koristis {}??

Data cine PROPS and STATE.

**PROPS ARE IMMUTABLE**. THEY CANNOT BE CHANGED. THEY ARE READ-ONLY. Ako hoces da promenis props, treba ti state. PROPS are object. Ako promenis nesto u dete komponenti, ta promena ce se odraziti na roditeljsku komponentu.

**ONE-WAY DATA FLOW**

DATA(Podaci) mogu se prenositi samo sa roditelja na dete. NE MOZE SA DETETA NA RODITELJA.

Angular ima two-way dataflow.

**THE RULES OF JSX**

JSX radi slicno kao HTML ali se moze ucu u JavaScript mode koriscenjem { }.

U JSX nisu dozvoljeni **STATEMENTS** (id, else, for, switch, while...)

JSX produces a JavaScript expression.

**DIFFERENCES BETWEEN JSX AND HTML!**

s

**46.RENDERING LISTS (MAP F-JA)**

To je kad imas niz i treba da kreiras po komponentu za svaki element niza. Koristis Map metodu.

ctrl + / AKO ZELIS DA ZAKOMENTARISES VISE LINIJE KODA. U JSX, komentari se pisu izmedju { }.

Znaci za svaki element niza, kreira se jedna komponenta i kao props se prenosi ceo pizza objekat.

  <div>

        {pizzaData.map((pizza) => (

          <Pizza pizzaObj={pizza} key={pizza.name} />

        ))}

      </div>

Svaki objekat mora da ima key props i da bude unikatan.

TAKODJE JE MNOGO BITNO DA POGODIS SEMANTICKE ELEMENTE

DA LI JE: <div>, <span>, <ul>, <li>, <p>, <h3>....

**47.CONDITIONAL RENDERING**

Rendering nekog dela UI na osnovu odredjenog uslova. Treba znas **SHORT CIRCUTING** sa && operatorom. Short circutin znaci da ako je value(isOpen) true vraca ono desno od && ako nije ne vraca nista. Kao isOpen NIKAD NE SME VREDNOST DA BUDE. Gledaj da uvek bude true ili false. Znaci ako je isOpen = true tada ce da se vrati ovaj <p>Open</p> ako nije true nece da se vrati.

{isOpen && <p>Open</p>} SHORT CIRCUTING

**48.TERNARIES CONDITIONAL RENDERING**

condition ? valueIfTrue : valueIfFalse

{props.isLoggedIn ? <h1>Welcome back!</h1> : <h1>Please sign up.</h1>}

U JavaScript mode, izmejdu { }, mora da se pise nesto sto GENERISE VREDNOST. A if else for while ne generisu vrednost, zato se ne mogu koristiti neko koristis ternaries ili conditional pristup. Jako bitno.

**49.CONDITIONAL RENDERING WITH MULTIPLE RETURNS**

React komponenta moze samo jedan blok ( ) da vrati, ali moguce je uvesti CONDITION tako da vrati blok u zavisnosti od nekog uslova. Komponenta vraca kao return JSX. Ali van return moguce je koristii if, for, while i ostalo.

 if (props.pizzaObj.soldOut) return null;

**50. EXTRACTING JSX INTO NEW COMPONENT**

Prettier nece da se pokrene ako naidje na greske neke. Mozes da pises komponentu unutar neke komponente ali se tako ne radi. Svaka komponenta se zasebno pise. Ako je potrebno neke podatke da joj prosledis, to radis preko props. Kad komponenta postaje prevelika, ti napravi neku manju i ubaci je u tu vecu komponentu.

**51. DESTRUCTURING PROPS**

Kako ne bi stalno pisao props. koristis destructuring.

**ctrl+D** -> Selection -> Add Next Occurence (kad imas vise neceg isto da obrises, KORISTI OVO)

Umesto props odmah koristis objekat koji si mu prosledio { pizzaObj }, ne mora kucas props.pizzaObj.podatak... Ne zaboravi samo viticaste zagrade. Sa viticastim zagradama destruktujes props.

**52. REACT FRAGMENTS <> </>**

Omogucava da se spoje neki elementi bez ostavljanja tragova u HTML stablu, DOM.

React fragment je <>. MANJE VECE. Znaci to omogucava da se wrapuju elementi, jer JSX podrzava samo jedan root element. I kako ne bi imao dodatni div koji u sebi ima dva elementa, mozes da koristis <> i da vrapujes ta dva elementa tako da budu kao jedan. WOW. Mozes da i pises <React.Fragment>.

**53. SETING CLASSES AND TEXT CONDITIONALLY**

Sto manje da se javlja kopiranje koda. Znaci ako imas neki uslov gde ako je true treba prikaze jedno a ako false treba prikaze drugo, mozes da koristis ternarne operatore unutar elemenata. Na primer:

 <span>{pizzaObj.soldOut ? "SOLD OUT" : pizzaObj.price}</span>

Ternarne operatore skoro uvek pises u { }.

 <li className={`pizza ${pizzaObj.soldOut ? "sold-out" : ""}`}>

KORISTE I TEMPLATE LITTERAL. Ovako je lepse. Kad ima razmak to znaci dve klase.

<li className={pizzaObj.soldOut ? "pizza sold-out" : "pizza "}>

**54. SECTION SUMMARY (Sta smo naucili?)**

**Komponente** su building blocks user interfejsa. Komponente se pisu koriscenjem JSX. Komponente se organizuju u componnet tree.

**Props** (skracenica od property) koriste se za prenos vrednosti izmedju komponenti. Prenose se samo u jednom smeru, odozgo nadole, sa roditelja na dete.

**Koriscenje MAP f-je** gde se prolazi kroz listu i za svaki objekat se kreira nova komponenta <Pizza /> i kao props joj se prosledjuje taj objekat

**55. CHALLENGE( neka kartica da se napravi, kasnije cu to :))**

**56. SECTION 6 OVERVIEW**

Prave se interaktivne komponente. Kako se hendluju eventi. States za updejtovanje UI. Kreiranje formi na “React nacin”, Controlled elements.

**57. LETS BUILD A STEPS COMPONENT**

App komponenta neka ti bude uvek root komponenta aplikacije.

Instaliraj ekstenziju -> Color Highlight -> Kako bi heks vrednost dobila boju koja je.

Ternarni operatori (condition ? true : false) se mnogo cesto koriste, treba budes dobar sa njima.

**58. HANDLING EVENTS IN REACT WAY**

U React-u se ne koriste selektori za selektovanje DOM elemenata niti se koriste eventListeneri. Koristi se nesto slicno. Za obradu pritiska na dugme koristi se **onClick** = { } prop. Moze i **onMouseEnter** = {}. Ne moze samo da stoji {alert(“Previous”)} jer ce da se izvrse pre klika na dugme, kad se kompajlira kod. Zato moras bar arrow f-je da napises.

 onClick={() => alert("Previous")}

**59. WHAT IS STATE IN REACT?**

State je ono sto cini komponentu interaktivon. Svaka promena nad state se reflektuje promenom nad UI.

State je najbitniji koncept u React-u. Sve se zasniva na njemu. Redux ima neke veze sa state.

State is data that component can have over time. State je memorija komponente. State sadrzi podatak koji ce se renderovati u komponenti. Svaka komponente treba da moze da drzi i prikazuje DATA tokom rada aplikacije. Svaki delic informacije je delic state-a. State promenljiva ili “piece of state” je jedna promenljiva unutar komponente.

Updejtovanje stejta trigeruje REACT da rerenderuje komponentu. Automatski ce kreirati novi component view. Kada je jedna komponenta renderovana, mi to zovemo VIEW. I kada se iskombinuju svi VIEW dobija se kompletan user interface. State omogucava da user interfejs bude sinhronizovan sa DATA. Promenis STATE => promenis UI. State je tool. State is most powerfull tool in React. Understanding mechanics of state will unlock the power of React development for me.

**60. CREATING A STATE VARIABLE WITH useState**

Klikom na dugme treba promenljiva step da se promeni u projektu 03. State omogucava da promenljiva postane dinamicna.

Ukucas useState kliknes enter i automatski se importuje ovo:

import { useState } from "react";

F-ja useState(1) kao parametar ima pocetnu vrednost promenljive step. useState vraca niz koji ima dve vrednosti. Prva vrednost je vrednost promenljive a druga vrednost je f-ja za manipulisanje vrednoscu promenljive. Mozes if da koristis, ne moras uvek ternarne operatore. If ne mozes d akoristis u JSX unutar Js moda { }. Tada uvek moras ternarne operatore da koristis.

useState je hook. Svaka f-ja koja pocinje sa **use** je hook. Hook-ove je jedino moguce pozvati unutar komponenti koje se exportuju. Ne mogu da se pozivaju unutar if statement, ili loop-a. NE MOGU SE POZIVATI UNUTAR DRUGIH F-JA. EsLint je kao kompajler, prijavljuje greske i warninge.

State jedino updejtujes sa seter f-jom (setStep). Nikako na drugaciji nacin.

const [step, setStep] = useState(1);

**61. DONT SET STATE MANUALY**

State jedino updejtuj koriscenjem seter f-je. Ovde pokazuje sta se desi ako probas da updejtujes state manuelno. Ako probas manuelno da updejtuse state sa step = step +1, React nikako ne moze da zakljuci da time treba da updejtuje state. Zato se nikakva promena nece desiti na UI. Zato je React obezbedio seter f-ju za updejtovanje state-a.

React is all about immutability. State jedino mozes da updejtujes alatima koje ti React daje. Uvek koristi const za definisanje state-a. Kao parametar u useState moze biti i objekat. Ako hoces da updejtujes bilo sta u state-u, uvek koristi seter f-ju. State uvek tretiraj kao immutabile, kao nesto sto ne mozes da izmenis direktno vec jedino koriscenjem odredjenog alata (setter f-ju)

62. THE MECHANICS OF STATE