Contents

[SECTION: 1-5 2](#_Toc153702879)

[SECTION 6: STATE, EVENTS, AND FORMS, INTERACTIVE COMPONENTS 16](#_Toc153702880)

[SECTION 7 : THINKING IN REACT 21](#_Toc153702881)

[SECTION 8: EAT-N-SPLIT APLIKACIJA 25](#_Toc153702882)

[SECTION 10: THINKING IN REACT: COMPONENTS, COMPOSITION AND REUSABILITY 26](#_Toc153702883)

[SECTION 11: HOW REACT WORKKS BEHIND THE SCENES 27](#_Toc153702884)

[SECTION 12: EFFECTS AND DATA FETCHING 27](#_Toc153702885)

[SECTION 13: CUSTOM HOOKS, REFS, AND MORE STATE 27](#_Toc153702886)

[SECTION 14: REACT BEFORE HOOKS: CLASS-BASED REACT 27](#_Toc153702887)

[SECTION 15: ADVANCED REACT + REDUX, 4 PROJECTS 27](#_Toc153702888)

[SECTION 16: THE ADVANCED useReducer HOOK 27](#_Toc153702889)

[SECTION 17. BUILDING SINGLE PAGE APPLICATION 27](#_Toc153702890)

# SECTION: 1-5

***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.

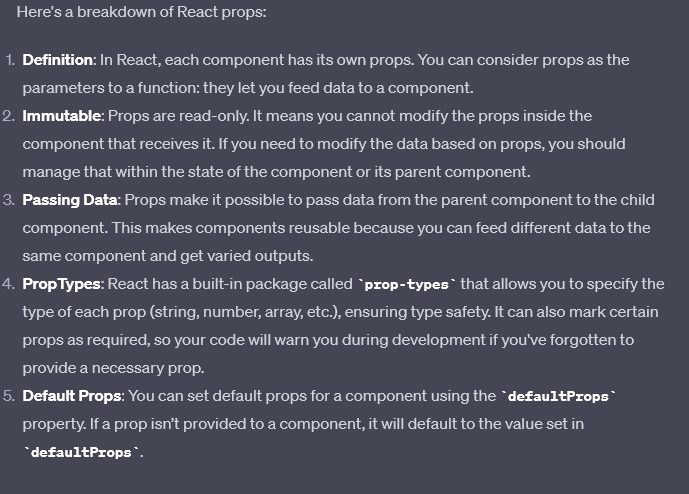
const [count, setCount] = useState(0);

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

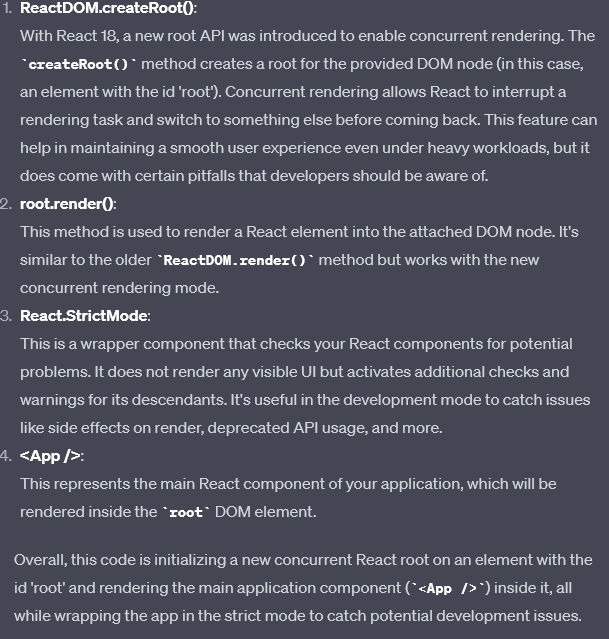
npm cache clean --force

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 :))**

# SECTION 6: STATE, EVENTS, AND FORMS, INTERACTIVE COMPONENTS

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 IN REACT**

U React-u se ne manipulise DOM-om direktno. Zbog toga sto je React deklerativan, nije imperativan. Nikad ne diras DOM u kodu. React updejtuje component view tako sto rerenduje celu komponentu svaki put kad se dese promene nad DATA. React sklanja stari view i postamlja novi view svaki put kad se desi neka promena. Kad je STATE updejtovan, komponenta se automatski rerenderuje. Svaki put kada zelis da updejtujes izgled komponente ti zapravo updejtujes njen state. A react reaktuje na updejt i vrsi rerendovanje komponente. Zbog toga se React zove React.

REACT REACTS TO STATE CHANGES BY RERENDERING USER INTERFACE. Frameworci postoje kako bi singronizovali UI sa DATA.

**63. ADDING ANOTHER PIECE OF STATE**

&times

Pravi iksic od button. State je memorija komponente. Cuva podatak tokom vremena. Koristi React fragment < > </>

**64. REACT DEVELOPER TOOLS**

React tim je napravio dev-tools specificne za React. Jako su korisne kada se radi sa state.

<https://react.dev/learn/react-developer-tools>

Treba skines ekstenziu za mozilu. Coomponents u dev-tools prikazuje component tree. Uz pomoc njega vidis kompletno component stablo svoje cele react aplikacije. Dev-tools ti pomaze da testiras aplikaciju sa razlicitim vrednostima statovima i tako to i da vidis component stablo.

**65. UPDATING STATE BASED ON CURRENT STATE**

Da se state updejtuje na osnovu trenutnog state-a. Znaci u seter f-ji kao parametar moras da prosledis f-ju. Bar arrow f-ju da prosledis. Pises callback f-je. To radis kada setujes state na osnovu trenutnog state-a. Ako samo prosledjujes vrednost onda ne treba callback f-ja. Preko callback f-ja je bezbedniji nacin da se updejtuje state.

setStep((s) => s - 1);

**66. MORE THOUGHTS ABOUT STATE + STATE GUIDLINES**

Svaka komponenta ima svoju verziju state-a. Svaka instanca komponente radi nezavisno jedna od druge. Promene nad state u jednom komponenti na utice na state u ostalim komponentama. State je izolovan unutar svake komponente. Ceo UI = f(state). UI je reprezentacija state-a. React aplikacija se zasniva na promenima state tokom vremena i rerenderovanja UI svaki put kad se state promeni. Sa state, UI je reflekcija promena DATA tokom vremena. Pravila:

1.) Kreiraj state variablu svaki put kada komponenta ima neki data koji treba da se pamti tokom vremena. I koja se menja tokom vremena. Promenljive koje be deviniso sa let ili var, u React koristis state za takve promenljive.

2.) Svaki put kad zelis nesto u komponenti da bude dinamcino, kreiraj state i updejtuj state kad ta stvar treba da se promeni.

3.) Svaki put kad zelis da promenis izgled komponenti, updejtuj njen state. View komponente je reflekcija promena state-a tokom vremena.

4.) Cesta greska je je da pocetnici koriste state za svaku promenljivu unutar komponente. Ali to nije neophodno. NE KORISTI STATE ZA PROMENLJIVE KOJE NE TRIGERUJU RERENDER. Jer dovodi do pada performansi. Mastering state is mot difficult, ljolj.

**67. A VANILLA JAVASCRIPT IMPLEMENTATION**

**68. CHALLENGE #1 DATE COUNTER V1**

Izbegavaj dupliranje koda.

Znaci cela komponenta se rerenderuje. Ne mora za svaku promenljivu da koristis state. Kad se promeni state neke druge promenljive, komponenta ce se rerenderovati i kod ce se izvrsiti tako da i promenljive koje nisu state da se njihove promene prikazu.

**69. NEW PROJECT-FAR AWAY(OVERVIEW)**

**70. BUILDING THE LAYOUT**

Za Emoji ima precica( WINDOWS + .)

U realnosti, pravis posebaan fajl za svaku komponentu koju imas.

**71. RENDERING THE ITEMS LIST**

Renderovanje listi je naslozeniji proces u React-u?

**72. BUILDING A FORM AND HANDLING SUBMISSIONS**

Forme su fundamentalne. To su delovi web app gde se nalaze polja za input i selektovanje nekih vrednosti itd..

Semanticki, kljucna rec <form> </form>

<select>

        {Array.from({ length: 20 }, (\_, i) => i + 1).map((num) => (

          <option value={num} key={num}>

            {num}

          </option>

        ))}

      </select>

Pravi select formu sa 20 optiona od 1-20. Prva f-ja pravi niz od 20 elemenata od 1-20 i onda sa map za svaki element kreira po jedan option koji ima vrednost num.

<form className="add-form" onSubmit={handleSubmit}>

Ovo cini da se stranica ponovo ucita sto nije dobro. Sa react treba da se spreci da se stranica ponovo ucitava.

 e.preventDefault();

Ovo mora ubacis u handleSubmit da se stranica ne bi refresovala ponovo. onSubmit omogucava da se f-ja pozove i sa enter i klikom na dugme dok sa onEvent je samo sa klikom na dugme.

**73. CONTROLLED ELEMENTS**

Fundamentalni React koncept. Koriste se kada radis sa formama.

Da bi radio sa useState mora da ga importujes, da ne bi pisao mehanicki kad ukucas useState klikni enter i ono ce automatski da se inkluduje.

Da bi koristio controlled elements, prvo napravis neki state i onda taj state koristis kao input polje.

Znaci fazon je da koristis state kao input polje.

<input

        type="text"

        placeholder="Item..."

        value={description}

        onChange={(e) => setDescription(e.target.value)}

      />

onChange znaci da svaki put kad se nesto ukuca, poziva se ova setDescription f-ja koja updejtuje vrednost state-a. value uvek mora da radi zajedno sa onChange. Znaci treba ga napravis tako da React bude zaduzen za vrednosti unutar elemenata a ne DOM. A React je zaduzen za to ako koristis states. e.target.value je uvek string. Koristis funckiju Number() kako bi ga konvertovao u broj.

Znaci kontroled elements koristis u 3 koraka.

1.Definisies state

2.Koristis state na input gde zelis da cuvas ulazne podatke, <input value={stateNeki} />

3.Updejtujes state promenljivu sa onChange handler, gde svaki put kad se desi neka promena se updejtuje state. Na taj nacin je React zaduzen za state i za ceo element koji ga koristi. I onda ta state promenljiva sadrzi input vrednost i ti je onda lako mozes koristiti za sta oces.

Sad je pitanje kako da podatke iz jedne komponente preneses na drugu komponentu. A props ne mozes da koristis jer se parametri prenose samo sa roditelja na dete a ne moze izmejdu razlicitih komponenti.

**74. State vs Props (DIFFERENCE)**

Koja je razlika izmedju state i props? State je internal data, podatak koji se pamti unutar komponente u kojoj je definisan. Dok sa druge strane, props je eksterni data. Data koju poseduje roditeljska komponenta. Props su kao funkcijski parametri. State je memorija komponente. Moze da cuve data tokom vremena. State moze biti updejtovan on strane same komponente, koja ce biti rerenderovana. Props su read-only. Ne moze biti modifikovana od strane komponente. Props su slicni kao f-jski parametri. State se moze prenositi kao prop. State se koriste da naprave komponente dinamicnom. Props se koriste kako mi modifikovali dete komponentu. Ovo je pitanje koje moze da se padne na intervju.

**75, 76 NEKI EXERCISE**

# SECTION 7 : THINKING IN REACT

**77. Section overview**

Start thinking in React. Sta je State managment? State promenljive? Kako se komunicira izmedju roditeljske i child komponente. Pravi se FAR AWAY aplikacija.

**78. WHAT IS THINKING IN REACT?**

Za izgradnju react aplikacija je potreban kompletno novi mind-set. Treba da imas u glavi predstavu kako da koristis sve alatke koje React pruza. Treba da razmisljas o state-transitions. Prvi korak je da UI pretvoris u komponente i kreiras component-tree. Moras da kompletno razumes state: Kada ga koristis, koji su tipovi, i gde da postavis svaki state. Takodje treba zna i data-flow. Komponente treba da budu reusable.

**79. FUNDAMENTALS OF STATE MANAGMENT**

State je najbitniji koncept u Reactu i njegovo upravljanje. U statu je data koja se menja tokom rada aplikacije i ona bese omogucava da stranica automatski primeni izmene. State managment je kada svakom state-u dajes komponentu. Mora znas koji state ces gde da stavis.

GLOBAL STATE I LOCAL STATE.

Svaki state je ili LOCAL ili GLOBAL. Local je kada se nalazi samo u jednoj komponenti ili u par child-a. Samo mali broj komponenti ima pristup state-u.

Global state, njemu moze pristupiti bilo koja komponenta i ona je deljena izmedju komponenti.

UVEK POCNI SA LOCAL STATE I PREBACI GA NA GLOBAL AKO TI STVARNO TREBA.

State ti treba kad pamtis neki podatak i treba da se zapitas da li se taj podatak menja vremenom. Svaki put kad se azurira state azurira se i komponenta. State je neka promenljiva i nju koristi neka komponenta. State promenljiva se moze preneti i na dete koristeci props.

**LIFTING STATE UP**. Kada sa deteta preneses state na roditelja.

**80. THINKING ABOUT STATE AND LIFTING STATE UP**

State uvek kooristis kada imas neki podatak da zapamtis. Zapitas se: When to create state and where to place it?? Treba da znas da radis sa immutable nizovima. Radi nesto dodaje iteme u niz u state.

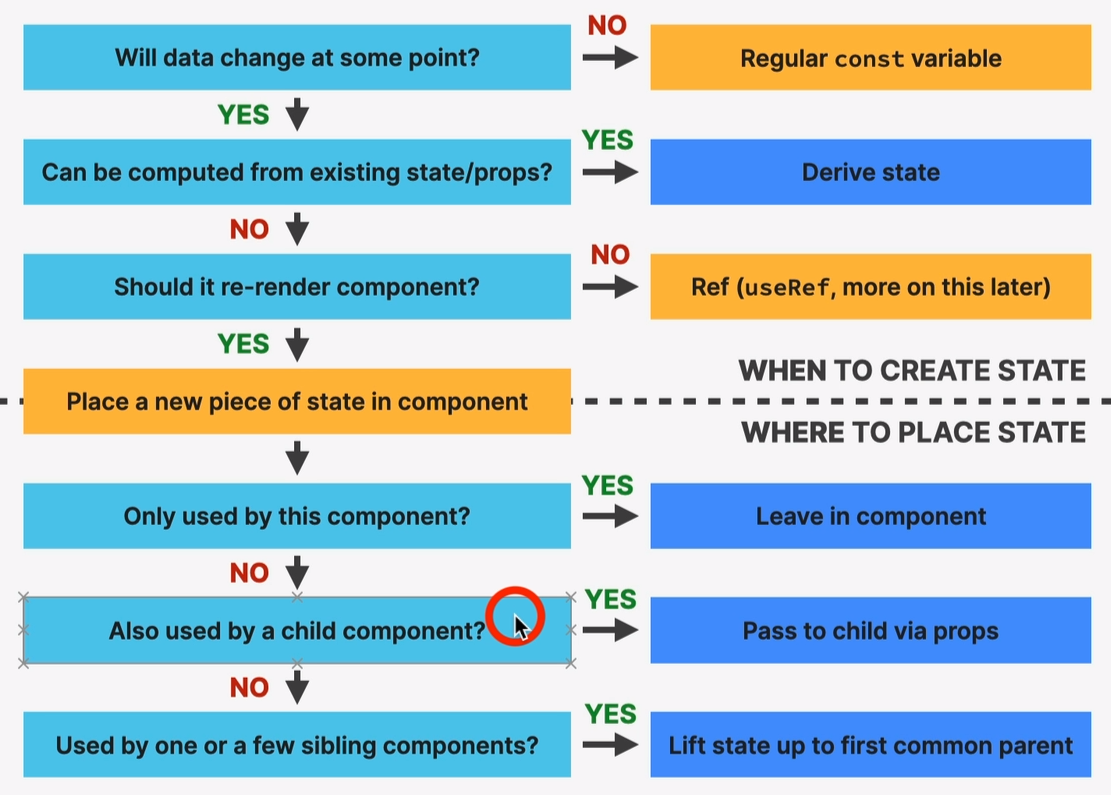
**LIFT UP STATE KONCEPT ISTRAZI!!**

{ITEMS}Tako ga destrukturujes valjda...

FILE-> PREFERENCES->THEME->COLORTHEME-> Visual studio light (NAJBOLJE ZA DIPLOMSKI KAD KOPIRAS KOD)

Kao props mozes da prosledis bilo sta, ukljucujuci i funkcije.

Znaci kada vise rodjaci komponenti hoce da koristi isti state, tada taj state LIFTUJES UP, do prve roditeljske komponente.



**81. LIFTING UP STATE**

Znaci state liftujes do prve roditeljske komponente. Na taj nacin svi potomci mu mogu pristupiti preko props-a. I tako mu veci broj komponenti moze pristupiti. Veoma je bitno da dobro razumes lifting-state. React je one-way data flow. Znaci kada liftujes state, od tad je state u roditeljskoj komponenti, nije vise u detetu. Props je read-only. Ne mogu se menjati.

**82. DELETING AN ITEM(CHILD TO PARENT COMMUNICATION)**

Brise neke iteme iz neke liste.

const [items, setItems] = useState([]);

function handleAddItmes(item){

setItmes((items) => [...items, item]);

}

Znaci ova addItems funkcija dodaje elementi iz json niza u state koji je neki niz. A mogo si i kao defaultnu vrenost u state da prosledis taj niz.

**83. UPDATING AN ITEM: COMPLEX IMMUTABLE DATA OPERATION**

Kako da dohhvati neku promenu nad input. Controlled element je kada se input vrsi u state.

<input type=”checkbox” value = {state.checked} onChange={()=> {}}/>

OnChange osluskuje kad se desi promena i poziva f-ju. Ta f-ja treba da updejtuje state.

function handleToggleItem(id) {

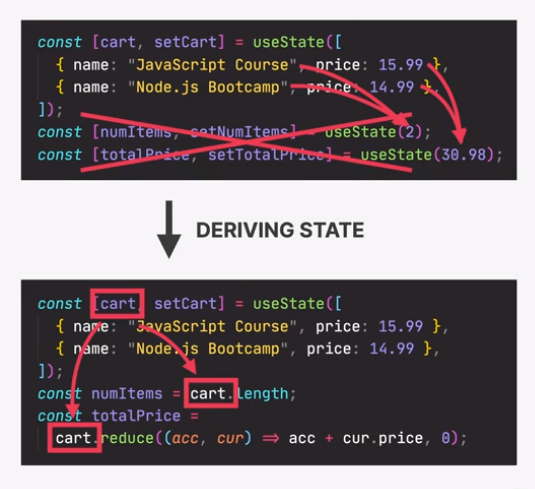
setItems((items) =>

items.map((item) =>

item.id === id ? { ...item, packed: !item.packed } : item ) );}

Ova f-ja poziva setItems metodu nad statetom items i azurira odredjeni item sa nekim id-jem.

**84. DERIVED STATE**

To je state koji je izracunat od strane nekog drugog state-a ili nekog propsa. ZNaci umesto da kreiras nove statove na osnovu postojeceg, mozes taj postojeci da uzmes i podatke izvuces i upises u neku obicnu promenljivu. Znaci to je proces izvodjenja podataka iz state-a. Znaci ne kreiraj dva state=a ako ti samo treba jedan.Znaci to treba da spreci nepotrebna rerenderovanja. Jer ako menjas dva state-a dva puta ce se renderovati, a ako menjas samo jedan, jednom ce se komponenta renderovati.

**85. CALCULATING STATISTICS AS DERIVED STATE**

Koristi dervied-state u primeru far-away

**86. SORTING ITEMS**

Implementira neko kompleksno sortiranje, ako ti ikad zatreba vrati se na ovu lekciju.

**87. CLEARING THE LIST**

Dodaje neko dugme gde brise listu item-a.

**88. Moving components into separate files**

Znaci svaka komponenta da bude u jednom fajlu. Tako se radi po firmama. Samo je komponente iskopirao svaku u svoj fajl.

**89. EXERCISE #1 Accordion Component(V1)**

//PRESKOCIO SAM

**90. THE CHILDREN PROP: MAKING REUSABLE BUTTON.**

Pravi reusable dugme. Svako reusable dugme treba da ima style, ono sto ce pise na dugme, i eventHandler. Znaci mozes da imas jednu komponentu dugme i da je vise puta koristis sa nekim razlicitim stilovima koje ces preneti kao props.

**CHILDREN** je ono sto stoji izmedju <Button> i </Button> Znaci ono sto pise na dugme. I onda to koristis kao {children} i prikazes ga na dugme. I tako mozes da pravis reusable dugme. Ovo se stalno koristi u react-u. Omogucava da komponente budu reusable. Znaci on prenosi ono sto se nalazi izmedju: <> i </>.

Uz pomoc ovoga, jedna komponenta se moze koristiti na razlicitim mestima. Pravis GENERICKE(reusable) komponente.

props={RGB(0,0,0)} key:value

export default function Button({style, onEvent, children}){

}

**91. MORE REUSABILITY WITH THE CHILDREN PROP**

Znaci radi jos neke primere sa {children} props.

Kad imas reusable komponentu da pravis, uvek treba iskoristis children prop mogucnost.

**92. 93. EXERCISES**

//OVO SAM PRESKOCIO

# SECTION 8: EAT-N-SPLIT APLIKACIJA

94.95.96.97 Prave UI aplikacije

# SECTION 10: THINKING IN REACT: COMPONENTS, COMPOSITION AND REUSABILITY

**109. COMPONENTS CATEGORIES**

Komponenta obicno spada u jednu od 3 kategorija.

1. Stateless component (prezentaciona komponenta) Moze preuzimati props, i obicno su to manje komponente koje se koriste samo za prikaz.
2. Statefull component- To su komponente koje imaju state. Medjutim i one mogu da budu reusable. To je kao neki search bar.
3. Structural component- To su neke vece komponente koji su kompozicija manjih komponenti. One su struktura web stranice.

Ne treba forsirati komponentu u neku od ove tri kategorije.

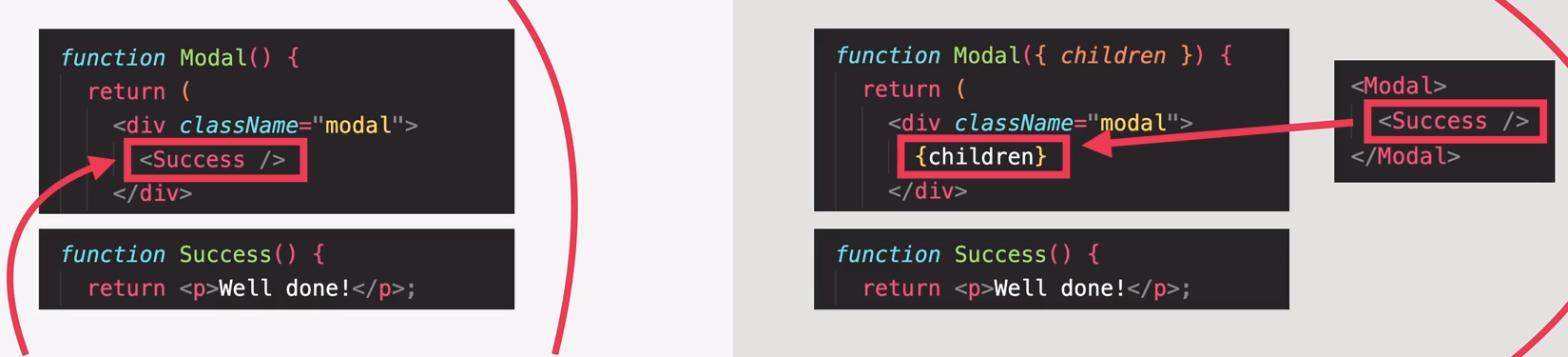
**110. PROP DRILLING**

Veliki problem. Znaci ako imas neki state koje dve duboko ugnjezdene komponente trebaju da koriste, lifting-up-state nije dobra opcija jer puno komponenti treba bespotrebno da dobije neki props i da ga prosledi dalje, sto nije dobro. Stvara se jako nepregledan i bespotreban kod.

**PROP DRILLING** to je kada neki props treba da prosledis kroz nekoliko child komponente koje ne koriste taj prop kako bi stiglo do duboko ugnjezdene komponente kojoj treba taj prop. I na kraju zavrsis sa mnogo propsa koji uopste nisu potrebni. Ima vise resenja, jedna od njih je component composition.

**111. COMPONENT COMPOSITION**

Kompozicija komponenti. Ja mislim da se ovo ne koristi toliko...To je umesto da komponentu nestujes unutar neke druge komponente, mozes da je prosledis kao {children} props. To omogucava visoku reusibilnost komponenti, da umesto komponentu da pises unutar komponenta, ti je prosledis kao children i ubacis je. Znaci koristis je za kreiranje highly-reusable komponente. Takodje resava props-drilling.

Component composition je tehnika kombinovanja komponenti koriscenjem children props-a.

# SECTION 11: HOW REACT WORKKS BEHIND THE SCENES

# SECTION 12: EFFECTS AND DATA FETCHING

**140. SECTION OVERVIEW**

95% React App fetchuje neki Data sa nekog API-a, cineci data-fetching neophodan skill za izgradnju web aplikacija. Jedan nacin za fetcovanje podataka je unutar effect-a. I ovo sekcija se time bavi. U ovoj sekciji se radi useEffect hook, kako se izvrsava, i kako mozemo Effect cleanup. Ucitavanje eksternih podataka u aplikaciju.

**141. THE COMPONENT LIFECYCLE**

Umesto component instance prica samo component. Znaci ovaj lifecycle govori o razlicitim ciklusima kroz koje komponenta moze proci tokom vremena.

**Mount/Initial render** (kad je komponenta prvi put renderovana, fresh state and props su kreirani, komponenta je rodjena u ovoj fazi.)

Nakon toga, komponenta moze biti **RE-RENDEOVANA** neogranicen broj puta, uvek kad je state promenjen. Komponenta ce se rerenderovati ako se: state promeni, props promeni, roditelj promeni ili context promeni. Ova faza je Optional.

Treca faza je UNMOUNT, to kada komponenta nije vise potrebna i kad se uklanja. Instanca komponente se uklanja sa UI interfejsa zajedno sa state and props. Znaci nova instanca se moze ubaciti kasnije ali je trenutna instanca obrisana.

Moze se napraviti da se kod izvrsava u specificnom trenutku u ova tri ciklusa i zato se koristi useEffect hook.

**142. HOW NOT TO FETCH DATA IN REACT**

Prvi put ucitava neke podatke u aplikaciju. To ce uraditi na pogresan nacin, znaci onako kako NE TREBA.

Ideja je da se fetchuju neki podaci kada se App komponenta renderuje po prvi put. Za fetcovanje koristi **omdbapi.com.** Treba dobijes neki Api key za dzabe. Kad to uzmes, odes na main-page i vidis kako koristis taj api. Kompletan data-request treba da se posalje na neki url. Koji kopiras i ubacis u fetch.

Znaci fetch metode, metode koje pribavljaju neke podatke od spolja, nikad ne treba da budu unutar render-logic, jer svaki put kad se rerenderuje komponenta, one ce se pozivati bespotrebno. Znaci svaki put kad se state promeni, rerenduje se komponenta, pa to rerenderovanje ponovo poziva fetch metodu pa se ponovo menja state, ponovo se rerenderuje i tako u krug dobijas po par 100 bespotrebnih fetcheva svaka sekunde i time opterecujes mrezu i servere. Znaci nikad fetch unutar state? Nikad ne setuj state u render-logic? ZNACI STATE I FETCH NIKAKO ZAJEDNO. Stvaraju infinite ciklus rendering-fetching. Zbog toga se koristi useEffect hook.

**143. useEffect TO THE RESCUE**

Ovde objasnjava jako vazan alat, a to je useEffect hook. Ideja ovog hooka je da nam omoguci mesto gde mozemo da pisemo side-effects, kao sto je fetch metoda. Znaci fetch unutar useEffect hooka ce biti izvrsen samo nakon initial render-a ili nekog drugog navedenog dogadjaja. Pise se slicno kao state. I mora ga importujes.

import {useEffetct, useState } from “react”;

useEffect(function() {

fetch(“URL”).then((res)=>res.json()).then((data)=>setState(data.Search))}, []

});

Funkcija je kod koji hocemo da registrujemo kao side-effect. Drugi argument je dependency-array. To je najzbunujuci deo ovog hook-a. Za pocetak prosledis prazan niz []. I onda se to poziva samo nakon MOUNT odnosno nakon initial-render. Znaci ovako sprecavas infinity-loop. Omogucavas da se podaci upisu u state bez stalnog rerenderovanja i fetchovanja. Znaci useEffect obicno sa fetch ide ruku pod ruku. Znaci sada se fetchuje jedino kad se komponenta mountuje. Ako hocemo da se podaci ucitaju kada se podaci loaduj-u.

RECAP: useEffect hook se koristi da registruje neki effect. Effect je ona funkcija koja sadrzi side effect koji zelimo da registrujemo. Odnosno effekat koji zelimo da se izvrsi nakon initial MOUNT-a. Prazan niz oznacava da ce se effect izvrsiti jedino nakon mount-a.

**144. A FIRST LOOK AT EFFECTS.**

Sta je effect? I koja je razlika u odnosu na eventHandler funkciju? Sta je side-effect? To je bilo koja interakcija izmedju komponente i spoljasnjeg sveta. Naprimer fetching data sa nekog API-a. Side effects ne smeju da se dese tokom renderovanja komponente. Side-effects mozemo kreirati na dva razlicita mesta u react-u. Prvi su eventHandleri, to su funkcije koje se trigeruju svkai put kada se desi dogadjaj koji osluskuju. Ali to cesto nije dovoljno da ispuni zahteve aplikacije. U nekim situacijama nam je potreban kod koji ce se izvrsiti odmah nakon INITIAL-RENDEROVANJA. To ostvarujemo uz useEffect hook. Omogucava nam da se kod izvrsava u razlicitim zivotnim ciklusima komponente. Kada se sta koristi:

EVENT-HANDLERS

-Kada fetchujemo data nakon nekog dogadjaja. Izvrsavaju se nakon nekog dogadjaja.

useEffect

-Nakon first-render.

Znaci obojca rade isto ali u razlicitom trenutku. Dependency array odredjuje trenutak kada ce se izvrsiti effect. Uz pomoc njega mozes reci kada zelis da se effect ponovo dogodi. Svaki effect moze pozvati i cleanup funkciju. Osnovni cilj je da komponenta bude sinhronizovana sa nekim externim dogadjajima i podacima. Koristimo useEffect da komponenta bude sinhronizovana sa eksternim podacima. Dok event-hendlere koristimo nakon nekog dogadjaja na komponenti i oni su “PREFFERED WAY OF CREATING SIDE-EFFECTS.”. Nemoj da overjuzujes useEffect hook. Sve sto moze preko eventHendlera bi trebalo tamo da bude.

**145. USING AN ASYNC FUNCTION**

Koristi neku asinhronu funkciju. Treba znas sta je async-await. Znaci pricao je i kako StrictMode dovodi do toga da se effect poziva dva puta..

**146. ADDNING A LOADING STATE**

Dodaje neki loading indikator u aplikaciju. Znaci hocemo da prikazemo neki loading-indikator dok se podaci ne ucitaju sa api-a. To je kada protekne neko vreme izmedju ucitavanja podataka i njihovog prikaza na UI usled spore internet konekcije. Kreira novi state koji se obicno zove isLoading. Ima vrednost true i false. Znaci svaki put kad se state promeni, komponenta se rerenderuje. Pozivas setLoading pre i posle fetcha. Pre fetcha ce se prikazati loader, nakon fetcha ce se prikazati ono sto treba.

{isLoading? <Loader /> : <OnoStoTreba />}

**147. HANDLING ERRORS**

Svaki put kad imas data fetching, uvek moras pretpostavis da moze da dodje do greske. Ovde prica kako se hendluju errors. Ukoliko korisnik izgubi internet konekciju, treba da mu se prikaze neka poruku, sta treba da se prikaze kad nastupi greska prilikom fetch-a.

Znaci ako !res nije dobar treba neku poruku da ispise. Ako nije uspeo da fetchuje, na UI treba da se prikaze poruka Failed to fetch. Ovde pise kod kako se to radi, jako jednostavan, ako treba pogledaj opet.

**148. THE useEffect Dependency Arrays**

Ovde objasnjava cemu sluzi Dependency Array u nutar useEffect hook-a. Bez njega react ne zna kada da pokrene effect. Po defaultu, efekt trepa da se pokrene nakon svakon renderovanja a to ne zelimo zbog infinite-loop, i zbog toga koristimo useEffect. Znaci svaki put kad se dependencie promeni, effect ce se izvrsiti ponovo. Svaka state promenljiva i props koji su korisceni unutar effecta MORAJU SE UKLJUCITI UNUTAR dependency array. Nepravilno njegovo navodjenje moze dovesti do “stale closure” problema.

useEffect hook se moze posmatrati kao neki event listener koji ceka da se neki od dependency promeni kako bi ponovo izvrsio odredjeni efekat. Znaci svaki put kad se promeni nesto unutar dependency array, efekat ce se ponovo izvrsiti. Efecti reaguju na state i props koji su korisceni unutar effekta. Effekti su reaktivni. Effekti se kao i States koriste kako bi aplikacija bila sinhronizovana. useEffect se koristi za sinhronizaciju nekog effekta sa statom aplikacije. Dependency je uvek state ili props. Mnogo bitno da ovo pogledas na slajdovima. Ako ne navedes dependency array, to znaci da se effekt pokrece nad svakim renderom sto nije dobro. Znaci effekt se pokrece tek nakon sto se komponenta iscrta na ekranu. Zbog toga sto su effect spori i da bi mogla komponenta da se lepo icrta prvo se icrta pa se onda effect poziva.

**149. SYNCHONIZING QUERIES WITH MOVIE DATA.**

Nesto menja u projektu. ZNACI KAD SE PROMENI NEKI STATE, POZIVA SE EFFECT.

useEffect(

function(){

console.log(“D”)}, [state]);

Znaci svaki put kad se promeni state, pozvace se effect koji ce stampani console.log(“D”); Ako se neki drugi state promeni, nece se izvrsiti ovo logovanje. Znaci useEffect osluskuje state, kad se state promeni, vrsi se pozivanje effecta. Effect ^^ State ruka pod ruku. Sinhronizovani.

**150. SELECTING A MOVIE**

Selektuje film kako bi ucitao nesto sa desne strane. Neke detalje o filmu prikazuje na desnoj strani . Treba ti state koji ce pamtiti koji film je selektovan.

# SECTION 13: CUSTOM HOOKS, REFS, AND MORE STATE

**159. SECTION OVERVIEW**

Ovde objasnjava detalje vezane za hooks. Hooks je lako nauciti ali tesko specijalizirati. Koja su pravila za hookove? Deep dive into useState. useRef. Custom hooks.

**160. REACT HOOKS AND THEIR RULES**

Sta su react hooks? To su specijalne funkcije unutar react-a koje nam omogucavaju da se hookujemo na neke unutrasnje mehanizme. Hooks su API-ji koji nam omogucavaju da koristimo neke funkcionalnosti react-a kao sto su stejtovi ili efekti. Hooks nam omogucavaju selekciju dom elemenata i jos puno toga. Svaki hook pocinje sa “use”. Mozemo kreirati i custom hooks. Omogucava nam non-visible logic reusability. Uz pomoc hookova, svaka komponenta moze imati svoj state i da pokrece neke efekte. React ima preko 20 buillt-in hookova.

useState, useEffect, useReducer, useContext, OVO SU NAJBITNIJI HOOKOVI.

Postoje dva najvaznija pravila za hookove koja moraju da se postuju. Hookovi jedino mogu da se zovu na top level? Nikako ih ne pozivaj unutar conditions ili loops ili nested functions. Hooks se uvek pozivaju istim redosledom. Drugo pravilo je da hooks se mogu pozivati jedino kroz komponente ili kroz custom hooks. EsLint sprecava da se ova pravila prekrse. Zasto hookovi moraju pri svakom renderovanju da se pozivaju istim redosledom. svaki put kad je aplikacija renderovana, react kreira react element tree ili ti neki virtualni dom. Takodje kreira i fiber tree gde je svaki element fiber. Svaki fiber sadrzi puno stvari. Takodje sadrzi listu hookova.

**161. THE RULES OF HOOKS IN PRACTICE**

Glavno pravilo da hookovi mogu da se pozivaju samo na top-lvl. Znaci ovde pravi primer kako bi pokazao da ona dva pravila moraju da se postoju kako bi radilo. Znaci hookove ne smes unutar IF, FOR, i takvih operatora da pises.m

//VECINU SAM PRESKOCIO

**162. MORE DETAILS OF USE STATE**

Ovde prica o nekim bitnims tvsrima vezane za useState jer je to najbitniji hook u react-u. Inicijalne vrednosti koje upisujes u state se koriste samo pri initial render, i nikad vise. Znaci u state se ispisalo pre nego sto se ucitao fetch. Zato prikazuje false u primeru. Znaci to je poenta, da se inicijalna vrednosti samo na pocetku koristi. I da je updejtovanje asinhrono i da nam treba callback funkcija ponekad. Osim sto se callback moze koristiti za azuriranje state-a, moze se koristiti i za updejtovanje state-a.

**163. INITIALIZING STATE WITH A CALLBACK (LAZY INITIAL STATE)**

//PRESKOCIO

**164. useState SUMMARY**

Sve sto smo do sada naucili na jednom slajdu. Znaci prvo ga koristis useState da kreiras state i imas f-ju koja se koristi za updejtovanje state-a. Pogledaj na prezentacija ovo lepo. Za apdejtovanja stejta nikad ne mutate objecte ili nizove nego ih zameni.

**165. HOW NOT TO SELECT DOM ELEMENTS IN REACT (refs)**

Kako da ne selektujes dom elemente u react-u. Zasto su nam potrebni refs u react-u. Kako da selektujemo element u react-u. Ne smes da koristis querySelector, nije react nacin da ga radis. React je deklerativan. Ne zelis da dodajes klase zbog selektovanja. Znaci zbog toga se koriste refs.

**166. INTRODUCING useRef HOOK**

Ovo je najlaksi hook od svih. Koristimo useRef da bi kreirali ref. Ref stands for reference, to je kao neka kutija u kojoj smestamo neki podatak koji zelimo da bude sacuvan izmedju renderovanja. .current propery, moze se menjati i citati iz njega. On je muttable, za razliku od ostalog u react-u. Njegov value ostaje isti izmedju renderovanja. Koristis refs da kreiras promenljive koje ce ostati iste izmedju renderovanja.

On se koristi za selektovanje i cuvanje dom elemenata. DOM element je isto neki data koji zelimo da selektujemo i cuvamo. refs je obicno za podatke koje se ne prikazuju na UI.

STATE VS REFS.

Refs su kao state sa manje moci. Oboje su persistentni izmedju renderovanja. Updejtovanje state ce izazvati rerenderovanje komponente dok updejtovanje refs nece.

**167. REFS TO SELECT DOM ELEMENTS**

Kroz primer pokazuje ovaj useRef. Koriscenje ref sa DOM elementom, odvija se u 3 koraka. Prvo kreiras ref, za to koristis useRef hook. Prenosis mu neku inicijalnu vrednost.

const inputEL = useRef(null);

ref={inputEl} //Ovo dodajes kao props u inout element

I sada je taj input element povezan sa refom. i nad inputEL.current.focus() mozes da pozivas sta hoces. Znaci ko da si ga dohvatio sa querySellector.

**168. REFS TO PERSIST DATA BETWEEN RENDERS**

//PRESKOCIO SAM SVE

Daje promenljivu koja je consistent izmedju renderovanje bez renderovanja....

**169.. WHAT ARE CUSTOM HOOKS? WHEN TO CREATE ONE?**

Prica o custom hookovima. Sta su to custom hooks i kad se kreiraju. Custom hooks are all about reusability. U reactu imamo dve stvari koje mozemo da reusujemo. UI i Logic. I to je sve. Ako zelima da reusujemo UI, koristimo komponentu. Ako hocemo da reusujemo LOGIC, moramo da se zapitamo da li ta logika sadrzi neki hook, ako ne sadrzi onda je obicna funkcija a ako sadrzi onda pravis custom hook. Custom hooks mogucava logic reusability. Custom hooks omogucava reusability non-visual logic. Jedan custom hook treba da ima samo 1 svrhu. Treba da radi samo jednu stvar. Custom hook mor abiti reusable i portable. Postoje puno custom-hook biblioteka koje mozes downloadovati i koristiti. Custom hook je neka javascript funkcija. Custom hook uvek vraca nesto. Moze se koristiti kada nesto fetchujes. I komponte su js funkcije koje uvek moraju da vracaju neki jsx. Custom hooks mora koristiti 1 ili vise react hookova. Ime funkcije mora da pocne sa “use”. Ako ne pocinje sa use, react ce ga tretirati kao najobicniju funkciju.

**170. CREATING OUR FIRST CUSTOM HOOK**

Kreira useMovies hook. Ima dve strategija za odredjivanje custom hook-a. Prva je da li zelimo da reusujemo neku non-visual logik. Ili ako hocemo neki deo de ekstrakujemo unutar nekog custom hook-a.

Dobro je sto moze fetch da se ubaci unutar nekog custom hook-a. Hookove kreiras kao zasebne fajlove zbog preglednosti. Najbolje se kkombinuje useState sa useEffect. Custom hook vraca promenljive koje ce se koristiti.

Znaci ovo je mnogo jaka stvar za vidljivost i citljivost koda, gde neku logiku mozes da ubacis u neki custom hook, i samo g apozoves i on ti vrati vrednosti koje ti treba i ti ih samo koristis posle. Vrlo lagano. Znaci custom hook mora koristiti bar jedan react-hook. Custom hooks su reusable JS funkcije sa hookovima.

export function useMovies(){}

**171. CREATING useLocalStorageState**

Veoma popularan hook koji omogucava rad sa local storage u aplikaciji. Ponasava se kao useState samo se state pamti u localni storage.

//OSTALO DA PRESLUSAM

**172. Creating useKey**

//OSTALO DA PRESLUSAM

**173. CHALLENGE #1: useGeolocate**

//OSTALO DA PRORADIM

# SECTION 14: REACT BEFORE HOOKS: CLASS-BASED REACT

# SECTION 15: ADVANCED REACT + REDUX, 4 PROJECTS

# SECTION 16: THE ADVANCED useReducer HOOK

# SECTION 17. BUILDING SINGLE PAGE APPLICATION

**203. Section overview**

Najbitnija 3rd party biblioteka je React Router koja se koristi u svim velikim aplikacijama. Sa react Router mozes da kreiras aplikacije sa vecem brojem stranica koje se brzo prikazuju i ucitavaju nativ brzinama. Kao da je desktop ili mobilna aplikacija. To su SPA aplikacije. Kroz ovu sekciju ce da pravi WorldWise SPA aplikaciju. Koristi i CSS module za stilovanje.

**204. Creating our first APP with VITE**

Ovde quick lookuje na projekat. Kreira aplikaciju sa VITE.

**npm create vite@latest** //Kako bi koristio zadnju verziju vite, on koristio vite@4

Vite je moderan build tool. Mozes razlicite framework-e da izaberes.

Onda mora da instaliras neke dependencije.

**npm install** //Instalira sve pakete

node\_modules sadrzi sve neophodne biblioteke

React ne uvodi u obzir strukturu fajla. Umesto index.js je main.jsx. isto umesto App.js je App.jsx. I nema junk fajlove.

**npm run dev** // Tako ga pokreces, nece sam da otvori tab u browseru, mora sam fa ga otvoris.

Problem je sto VITE nema **ESLint**, mora ga sam ubacis. Bez njega lako mozes da napravis greske. OBAVEZAN JE. Ovo otkucaj da ga instaliras.

npm install eslint vite-plugin-eslint eslint-config-react-app –save-dev

Treba kreiras fajl: .eslintrc.json i u njega upises

{

“extends”: “react-app”

}

U vite.config.js treba da ga konfigurujes, dodas eslint plugin.

import eslint from “vite-plugin-eslint”;

plugins: [react(), eslint()]

Nakon ovoga si setupovao projekat.

**205. ROUTING AND SINGLE-PAGE-APPLICATIONS**

Ovde objasnjava sta su ustvari SPA aplikacije.

Sa ROUTING, on povezuje razlicite URL-ove sa razlicitim UI interfejsima. Svaki url metchujes sa specificnom React komponentom, i to je ROUTE. I onda kada se specificni URL poseti, specificna React komponenta ce biti renderovana.

Na primer, imas /home komponentu, /login komponenta, ili /app komponenta. Imas tri rute koje korisnik moze da poseti. Ovo omogucava laku navigaciju kroz stranicu. Routing ovogucava da korisnicki interfejs bude sinhronizovan sa trenutnim URL-om. U Reactu, Routing se ostvarruje koriscenjem 3rd party paketa React Router. I ovo je najkoriscenija 3rd party biblioteka na svetu. ROUTING je osnova za SPA aplikacije. Znaci za svaki URL se odredjena komponenta renderuje i na taj nacin ne moras da otvaras novu html stranicu nego na vec postojecu samo izrenderujes novu komponentu.

SPA aplikacije su aplikacije koje se izvrsavaju na browseru klijenta i oslanjaju se na ROUTING. Svaki put kada korisnik klikne na neki link, url u browseru se promeni, onda promena URL triggeruje da se DOM updejtuje. U SPA uvek ce javascript da updejtuje DOM. Obicno kad se klikne na link otvara se nova html stranica dok kod SPA stranica se updejtuje uz javascript. Nece nova html stranica da se ucitava. Sve se izvrsava na jednoj html stranici. I to joj daje dobar user experience. Svaki put kad se url promeni, vrsi se rerenderovanje komponente. Svaki klik menja URL i komponentu, i sve to bez reloadovanje stranice. Kad treba neki podaci oni se ucitavaju sa servera bez osvezavanje stranice. Znaci routing je jako dobra stvar.

**206. IMPLEMENTING MAIN PAGES AND ROUTES.**

Koristi React Route bliblioteku po prvi put. Sa routing povezujes specifican url sa specificnom komponentom.

npm install react-router-dom //OVO SAMO KAD GA PRAVIS KROZ VITE

Postoje dva nacina za deklarisanje ruta.

<BrowserRouter>

      <Routes>

        <Route path="/" element={<Home />} />

        <Route path="login" element={<Login />} />

        <Route path="contact" element={<Contact />} />

      </Routes>

    </BrowserRouter>

Znaci u ovaj section pravi rute...

**207. LINKING BETWEEN RUTES WITH LINK**

Povezuje rute sa linkovima. Ne zelis stranica da se refresuje kada kliknes link. Zelis dom da se rerenderuje a ne da se stranica ucita. Koristis Link element. Uz Link kad klikne na link on ce da se prabaci na to bez ucitavanje stranice. Treba d aga importujes.

import { Link } from "react-router-dom";

Sa **rfc + Enter** pravi ti templejt za komponentu. Mnogo jaka stvar.

Pravis posebno folder za komponente posebno za pages. I svaki page treba da ima nav bar da bi sve radilo kako treba.

Treba da highlaghtujes koju stranicu trenutno posecujes. Umesto link mozes NavLink element da koristis. Uz NavLink dobijas class=”active” i uz pomoc toga mozes da koristis za stilovanje ili svasta nesto. Znaci to je razlika izmedju NavLink i obican Link. Razlika je class “active”.

**208. STYLING OPTIONS FOR REACT APPLICATIONS**

Bitna stvar kod React aplikacija je njihovo stilizovanje koriscenjem css-a. Ovde koristi css module. Ovde objasnjuje koje sve opcije za stilizovanje imas. Razliciti nacini postoje za stilizovanje react aplikacije. React doesnt care about styling. Mozes da koristis:

1. Incline CSS local-scope, primenjuje se samo da taj element
2. CSS or Sass file stilovi zu globalni, svi elementi sa klasom moze da koriste te elemente. Problem u velikim projektima zbog className. Oni su nightmare za velike app. Css nikad nije globalan nego treba da bude scoped unutar komponente.
3. CSS Modules veoma slicni kao .css fajl samo gde se po jedan css fajl pravi za svaku komponentu. Fajl moze da scope samo unutar jedne komponente. I to cini komponentu “reusable”. To ce da koristi u ovom projektu.
4. Css-In-Js css mozes da pises unutar js fajla. Ono sto je specijalno je da omogucava da kreiras komponente sa stilovima koji su vec primeneni na nju.
5. Css framework kao sto je TAILWIND, kod njega koristis predifined stilove. Sve unutar JSX-a.
6. Imas i css biblioteke, komponent library.

**209. USING CSS MODULES.**

Objasnjava sve vezano za css modules. Nista ne treba da se instalira. Kreiras po .css fajl po komponenti. Treba da vodis racuna o konvenciji. Da se Zove npr: PageNav.module.css. Sve navodis kroz className. Ne mozes samo element da navedes. Stilovi moraju samo na tu komponentu da se primenjuju. className su uvek random i unikatni. Nikad nece dva ista className da se generisu.

<nav className={styles.nav}>

**211 BUILDING APP LAYOUT**

**212. NESTED ROUTES AND INDEXED ROUTES**

Trebaju nam nested rutes kada zelimo da deo korisnickog interfejsa bude kontrolisan od strane dela url-a. Nested routes je kada se deo UI renderuje na osnovu nekog dela URL-a.

Nested routes odredjuje koja komponenta ce se renderovati unutar neke vece komponente. Znaci to je u sustini ruta unutar rute. Imas rutu /app i u ju /app/cities gde ti lista gradove ili /app/countries gde ti lista zemlje. Nested rute deklarises unutar Route elementa. U sustini kreiras dete rutu unutar roditeljske rute. element ne mora da bude komponenta, moze obican jsx element kao na primer <p>.

Medjutim sad je problem kako ce ta komponenta da se prikaze unutar neke druge komponente.

Imas <Outlet /> element. Unutar njega se renderuju te komponente.. ahaaa, i treba da ga ukljucis.

import { Outlet } from “react-router-dom”;

U <Outler /> se renderuje ta Nested Route komponenta.

A idexed ruta je defaultna Child ruta koja ce biti prikazana ako ni jedan url se ne mechuje. Znaci kad samo imas /app, indexed ruta ce se prikazati.

<Route index element={<Homepage />} />

ZNaci svaki put kad se promeni ruta, promeni se prikaz na korisnickom interfejsu.

useState hook...

**213. IMPLEMENTING THE CITIES LIST**

Treba instaliras json server kako bi mogao da zoves neke APi-je.

npm i json-server

Ovde implementira CityList.

u package.json treba dodas: “server”: “json-server –watch data/cities.json –port 9000 –delay 500”

npm run server

I onda mozes da koristis fetch metodu. On poziva neki fejk api.

Znaci ovde ucitava neki json fajl, vadi podatke iz njega i povezuje na komponentu...

Treba da napravim fake json fajl...

**214. IMPLENETING THE COUNTRIES LIST.**

Ovde pravi listu zemalja. Snaci primenjuje one .map, .reduce operacije nad nizovima.. Znaci iz JSON, obilazi ga i vadi podatke i vraca niz... znas vec..

**215. STORING STATE IN THE URL**

State se pamti u url kako bi mogao da se koristi na ostalim mestima. Url je odlicno mesto da pamtis state. Narocito UI state. State u URL je easy accessible. Sve komponente mu mogu pristupiti. Znaci neki state mogu se smestiti u URL. I to je dobar nacin da se podaci premestaju sa stranice na stranicu. Sve je to moguce us React Router.

Za pamcenje state unutar url-a, koriste se params ili-ti query string-ovi.

Ovo su bas napredne stvari...

**216. DYNAMIC ROUTEW WITH URL PARAMETERS**

//NISAM GLEDAO

**217. READING AND SETTING A QUERY STRING**

//NISAM GLEDAO

**218. Programmatic Navigation with useNavigate**

Ovde useNavigate objasnjava.. to je neki custom hook. Programmatic navigation znaci da je moguce preci na nov URL bez da korisnik klikne na bilo koji link. I to je dobro nakon submiting form. Znaci kada korisnik popuni formu, zelimo da se automatski izvrsi prikaz nove strane, bez potrebe da klikne neki link.

Znaci ima neka forma koja treba automatski da se ucita.

useNavigate hook.

const navigate = useNavigate();

navigate(“form”)

Znaci nakon nekon eventa poziva se ovo navigate koji renderuje form bez klika na neki link... us useNavigate mozemo da navigujemo na neki url bez klika na link.

Znaci ceo fazon da se menja URL, da se rerenderuje UI, bez klika na link. I takodje moze da navigatuje unazad.

Kreira reusable button komponentu. Ima neke props ulazne parametre.

**219. PROGRAMMATIC NAVIGATION WITH <Navigate />**

Znaci imas i Navigate komponentu. Retko se koristi. Ali i dalje ima primenu unutar Nested Ruta. Isto se koristi za redirektovanje. U sledecoj sekciji objasnjava context API.

# SECTION 18- THE CONEXT API- ADVANCED STATE MANAGMENT

**222. WHAT IS CONTEXT API?**

Sta je Context API i koje probleme resava? Znaci kada je potrebno da neki state prosledimo u duboko nested child komponentama. **Prop-Drilling**. Prosledjivanje props-a kroz stablo nije dobro resenje. Mnogo je bolje resenje kada bi smo mogli neki state direktno da prosledimo sa roditeljske komponente na neku duboko-nested child komponentu. Context API omogucava da bilo kojoj komponenti da procita state bilo gde u stablu. Context API je sistem za prenos podataka kroz aplikaciju bez da se taj podatak proslejduje kao props kroz stablo. Znaci omogucava prenos podataka izmejdu komponenti bez koriscenja props-a.

Provider: Je komponenta koja daje pristup promenljivi. To je vrednost koju zelimo da brodcastujemo. Ta promenljiva ima vise state vrednosti.

Consmer: Komponenta koja cita. Ona se subskrajbuje na context. Mozes neogranicen broj consumera na jedan context. Svaki put kad je context updejtovan, svi konzumeri ce se rerenderovati zajedno sa provajderom.

Omogucava da se definise GLOBAL\_STATE.

**223. CREATING AND PROVIDING A CONTEXT**

Kreiranje contexta. pravis createContext(); On vraca context. Znaci ovo se radi iz 3 koraka.

1.Prvi korak je da kreiras novi context. PostContext je neka komponenta.

const PostContext = createContext();

2.Provide value to child components, kreiras po context za svaki state. Pravis Provajdera.

<PostContext.Provider

  value={{

    //Kreiras properti koji sadrzi po value koji zelis da prenes

    posts: searchedPosts, //ovo je state

    onAddPost: handelAddPost, //ovo je funkcija

    onClearPosts: handleClearPosts,

    searchQuery,

    setSearchQUery,

  }}

></PostContext.Provider>;

Znaci i funkcije mozes da prenosis kao props...

useContext hook.

Znaci context Receivere pravis unutar child komponente. Samo pokupis vrednosti iz contexta.

const {onClearPosts} = useContext(PostContext);

225. ADVANCED PATTERN: A CUSTOM PROVIDER AND HOOK

Contexte pises kao zasebne fajlove. Pogledaj te ebane call-back f-je unutar state-a zbog toga sto je state asinhron..

**238. ADDING FAKE AUTHENTICATION**

Obicno se authentikacija odvija u tri koraka. Prvo uzmemo email i password sa login forme i proverimo sa API-end-point da li sifra za datog korisnika odgovara. Da li je dobra. Ako je sve okej, redirektujemo korisnika da glavnu aplikaciju i pamtimo korisnika u neki state. I kao treci korak treba da zastitimo aplikaciju od neovlascenog pristupa, odnosno od korisnika koji nisu logovani. I on ce u naredne tri lekcije da implementira ova tri koraka.

E ovo fake, to se odnosi na prvi korak, nece da proverava kroz API vec ce da koristi neki hard-codded objekat i kroz njega ce da proveri da li je user i password okej. Kasnije ce da pravi aplikaciju gde podaci stizu iz prave baze od strane pravih korisnika.

U state pamti da li je user logovan ili nije i takodje pamti i korisnika.

Pamti neki context, FakeAAuthContext. Sta su contexti??

Znaci kad se klikne na login dugme, poziva login f-ju u kojoj se proverava da li je korisnik ispravno uneo podatke.

# SECTION 29: IMPLEMENTING MORE FEATURES:

**374.Section overview**

Dodaje razlicite funkcionalnosti.

**375. Client-Seide Filtering: Fitlering Cabins**

Pravi reusable komponentu za filtriranje.

# SECTION 30: DEPLOYMENT WITH NETLIFY AND VERCEL

# SECTION 31: THE End!