ReactJS – Jawad Samad:

Hierbij een korte uitleg over de redenen waarom ik voor bepaalde React functies heb gekozen. Allereerst wil ik er even bij vertellen dat ik het SweetCoffeeMock.js NIET heb gebruikt. Ik heb Erik Mols op 3 februari een mail verstuurd om meer uitleg te geven over het gebruik van de SweetCoffeeMock.js bestand, maar helaas heb ik daar geen antwoord op gekregen en kon ook geen antwoord via iemand anders krijgen, omdat wij geen 2de docent aangewezen hebben gekregen die hier uitleg over kon geven.

Uitleg SweetCoffeeMock.js bestand.

Jawad Samad <jawadsamad@gmail.com> ma 3 feb. 18:35

aan epmols, Birgitmaas, remi.p, martijnterstappen, cvessen

Beste Erik,

Wij vroegen ons eigen af wat nou eigenlijk de bedoeling is van SweetCoffeeMock.js bestand dat meegeleverd is bij de eindopdracht? Wij hebben hierover geen uitleg gekregen, zouden wij hier alsjeblieft wat meer toelichting over kunnen krijgen?

Wij horen het graag.

Met vriendelijke groet,

Jawad Samad

Mijn versie is voornamelijk uitgewerkt in React-Hooks. Ik kan het ook eenvoudig omzetten naar React class based componenten, maar heb bewust gekozen voor Hooks omdat het gebruik van functionele componenten “tegenwoordig” bijna net zo krachtig is als het gebruik van class based componenten. Ik zou voor class based componenten hebben gekozen als ik meer met lifecycles te maken had. Enkel zag ik er bij dit eindopdracht niet het nut van in om te gaan werken met lifecycles:

Ik heb 3 folders:

- Componenten (opslag van alle componenten met ieder zijn eigen folder met eventueel een css bestand)
- Context (2 context bestanden, voor het aflezen van de globale waardes)
- Images (1 afbeelding, achtergrond afbeelding)

Ik ben begonnen met het creëren van een button component:
<Button /> - in dit bestand heb ik het heel simpel en basaal gehouden. Alle informatie komt bij dit component binnen via props. Ik heb alles gedestructureerd om het gebruik van props… te minimaliseren. Dit component bevat 1 functie en dat is het uitvoeren van de “handleClick” methode, waarbij ik in de handle clickmethode vermeld dat de “clicked” methode uitgevoerd moet worden van mijn App.js bestand.

App.js

In mijn App.js bestand (regel 13-14) verwijs ik door naar het DataContext en ErrorMessage bestand. In mijn DataContext.js en ErrorMessage.js bestand staan globale waardes met defaultvalues. Vervolgens staan er 2 useStates, regel 17 en regel 21.

Regel 17 van App.js: useState is voor het zichtbaar maken om wat voor tekst er vertoond moeten worden in de <Statusbar /> component. Mijn initiële waarde is “Maak uw keuze” en vervolgens set ik de waarde in de functie op regel 23. De functie ontvangt een “nameValue” van de <Button /> component(zie <Button /> component waarbij de “handleClick functie van de button component de naam van de button terug geeft aan de prepared functie”). Zodra de naam ontvangen is vanuit de <Button /> component, wordt de state van de <Statusbar /> component geset op “… wordt voorbereid”. Hierna heb ik 2 setTimeOut functies erbij gezet, waardoor na iedere paar seconden (6-8 seconden) de state wederom veranderd naar “… is klaar” en weer terug naar de originele value “Maak uw keuze”.

Regel 21 van App.js: useState is voor het disablen van mijn <Button /> componenten. Hierbij heb ik gebruik gemaakt van de globale waardes van DataContext.js, waarbij ik telkens de waarde van de globale state verander. Initiële waarde van de <Button disabled={false}/> is false. Dus de button is niet disabled, maar zodra de <Button /> geklikt wordt, dan komt er een timeout van 7 seconden voordat de button weer word enabled.

Regel 35 van App.js: Hier heb ik 6 buttons gecreëerd waar ik ook de props in vermeld.
Props:

- Disabled = waarde van disabled komt van DataContext.js en manipuleer ik via useState zoals hierboven is uitgelegd.
- Clicked = functie prepared roep ik op met daarin de setTimeOuts en de setStates
- Name = Ik geef hier de naam door
- Children = Titel van de buttons

Voor mijn 6 buttons voer ik ook conditionele rendering toe. Dus hierbij kijk ik of in de DataContext.js bestand de waarde van de sugar, milk of chocolate op true staat. Als dit op true staat, zijn de buttons niet gedisabled en werken ze naar behoren. Enkel als je dan in mijn DataContext.js bestand de waardes van sugar, milk of chocolate op false zet, dan zijn de respectievelijke buttons ook disabled en zijn ze ook niet klikbaar(Click props heb ik een anonieme functie in geplaatst, anders kreeg ik telkens de foutmelding dat er geen Click methode bestaat).

Regel 50 van App.js : Staat de <Statusbar /> component dat enkel de useState afleest zoals eerder omschreven. Niet heel veel spectaculairs te beleven in de <Statusbar /> component. Enkel dat het telkens de state veranderd en daarmee ook de tekst.

Regel 51/52 van App.js: Hier heb ik de <Slider /> componenten geplaatst. Wederom wordt dit conditioneel gecontroleerd of de waardes sugar of milk uit de DataContext.js bestand op true staan of op false. Als ze op true staan, dan werken ze naar behoren. Mochten ze op false staan, dan zijn de sliders gedisabled en wordt de tekst zichtbaar gemaakt dat de gebruiker suiker of melk moet bij vullen.

*Slider component:
Regel 7 van Slider.js: Hier heb ik wederom een instantie gemaakt van de DataContext.js bestand om de globale waarde af te lezen. De waarde van value (regel 9) zet ik dan ook gelijk aan de waarde van dataContext.value. De reden hiervoor staat beschreven in regel 11, waarbij ik een useState gebruik om de waarde te manipuleren. Waarde staat initieel op 0, maar door de functie op regel 14 wordt de waarde d.m.v. useState gewijzigd tot waar de gebruiker zijn muisklik verschuift. Vervolgens roep ik een andere functie op van regel 19 waarbij door een setTimeOut de waarde van de slider weer op 0 wordt gezet na een aantal seconden. In de return heb ik gebruik gemaakt van <input />, waarbij ik de onChange waarde zet op de functie handleChange. Disabled verwijs ik door naar props, waarbij er in App.js props naar DataContext.js wordt gekeken of de waardes sugar of milk op false of op true staan.

Regel 54 van App.js: staat de <Error /> component. Ik heb het op dit moment als commentaar erin verwerkt. Maar als je gebruik ervan wilt maken, dan kun je het simpel weg uncommenten en in de <Error error={error1 / error2/ error3}/> kan je de foutmeldingen doorgeven. Error component is ook niet heel uitgebreid. Hier staan enkel de foutmeldingen dat de gebruiker krijgt als de errorwaardes 1-2 of 3 vertoond worden met ieder zijn eigen foutmelding. Alleen moet ik er wel bij vermelden dat ik de initiële waardes voornamelijk aflees van ErrorMessage.js bestand wat een context is en de globale waardes heeft.

Testen:
Wat testen betreft..

https://edhub.novi.nl/study/courses/158/content/2639. Dat is alles wat aangeboden wordt vanuit edhub. Niet heel uitgebreid, kan ook aan mij liggen.