Verantwoordingsdocument

React Sweet Coffee interface – Joris van Laar

Gebruikte libraries

Voor de ontwikkeling van de SupplementSlider heb ik gebruikgemaakt van de React UI library van Ant Design: https://www.npmjs.com/package/antd
Omdat Ant Design nog moet migreren naar de nieuwe React lifecycle methods onderdruk ik een deprecation warning die wordt getriggered door de animatie van de slider, zie lines 7-11 van index.js
Meer info hierover op: https://github.com/ant-design/ant-design/issues/9792
Het npm commando voor het installeren van de Ant Design library is: npm install antd

Voor het testen heb ik gebruikgemaakt van Jest en Enzyme. Hoewel waarschijnlijk onnodig, heb ik voor de zekerheid handmatig Jest geïnstalleerd middels het npm commando: npm install --save-dev jest

Enzyme heb ik geïnstalleerd met het npm commando: npm install --save-dev enzyme jest-enzyme enzyme-adapter-react-16

Use Case 1: Selecteren van drank

1. Algemene opzet

De App component vormt voor mijn interface het centrale punt dat communiceert met de besturingscomputer (SweetCoffeeMock) door binnen App een instance aan te maken van de mock. Zodoende kan de functionaliteit die binnen de mock leeft worden aangeroepen vanuit App.

Tegelijkertijd is App ook het parent component die de verschillende interface componenten samenbrengt. De state van App is allesoverheersend en is bepalend voor het gedrag van de interface. Met behulp van het doorgeven van props is App in staat informatie te delen met de verschillende componenten.

De App component is parent van de volgende child components:

- CoffeeButton
- SupplementSlider
- ErrorButton
- ErrorOverlay

2. Layout van de interface

Binnen App plaats ik de verschillende componenten in een html table, zodat ik controle heb over de positionering van de componenten.
De resolutie van de interface staat vast (1280x800 pixels) en is niet responsive opgezet om mee te schalen met andere resoluties. Elk component heeft een eigen, aparte css file. Daarnaast heb ik zoveel mogelijk inline styling proberen te vermijden, omdat ik dat nadelig vind voor de leesbaarheid van de code. Het enige moment waarbij ik wel gebruik maak van inline styling is om dynamisch de kleur van de CoffeeButton en de ErrorButton aan te passen, wanneer deze disabled zijn. Zie CoffeeButton.js -> line 15-18 en ErrorButton.js -> line 15-18

3. Hoofdscenario

Binnen App initialiseer ik in de constructor functie eerst de state van de applicatie, waardoor alle interface elementen bij opstart enabled zijn en de status op “Ready for use” staat

In CoffeeButton.js is de onClick property van de CoffeeButton gelinkt aan de makeCoffee prop uit App.js. Afhankelijk van op welke knop wordt geklikt, wordt een specifieke ‘koffie-maak’ functie in App gecalled. Wordt er bijvoorbeeld op de Cappuccino knop geklikt, dan wordt de makeCappuccino functie in App gecalled. Deze functie past de state van de interface op de volgende punten aan:

- Statusbericht verandert in “Making Cappuccino…”
- Alle interface elementen zijn disabled

Naast deze aanpassing in de state wordt de makeCappuccino functie van de mock gecalled, die in dit geval een simpele time-out functie is ter simulatie, om vervolgens na uitvoering hiervan de callback functie setToReady in App te callen. setToReady reset als het ware weer de state van de interface, waardoor de status message weer verandert in “Ready for use” en alle interface elementen weer enabled zijn.

4. Alternatieve scenario’s

Ik heb ervoor gekozen om drie sliders voor de supplementen aan te maken, naast “sugar” en “milk” heb ik een extra “cacao” slider aangemaakt. Hoewel deze niet in de briefing voorkwam leek mij dit de meest logische oplossing om aan de eisen van Alternatief Scenario 3 te voldoen.

Daarnaast heb ik in de mock de argumenten voor de verschillende drank-functies aangepast. Alleen de dranken die disabled raken als een bepaald supplement op 0 staat, ontvangen de waarde(s) van die desbetreffende supplementen als argumenten. makeBlackTea heeft dus bijv. geen argumenten, terwijl makeChoco wel een argument ‘cacao’ heeft. Hoewel dit misschien niet strookt met de werkelijke bereiding van alle dranken, leek dit me de meest duidelijke toepassing o.b.v. de briefing.

Binnen SupplementSlider.js wordt een onChange property bijgehouden die de waarde van de slider registreert. Die waarde geef ik vervolgens door aan de App m.b.v. de updateSupplement prop, die de sugarValue/milkValue/cacaoValue attributen van de state set. Omdat setState een asynchrone method is en dus niet altijd meteen een component update, moest ik gebruikmaken van async/await in de

updateSugar/updateMilk/updateCacao functies in App, om ervoor te zorgen dat de value die in de state van App wordt geset, synchroon is met de slider.

Binnen de updateSugar/updatMilk/updateCacao functies in App bouw ik ook een conditie in dat als de value van de slider nul is, de Cappuccino en/of Chocolate & Wiener Melange buttons disabled zijn. Specifiek hiervoor heb ik ook de attributen 'capoDisabled' en 'chocoAndWienerDisabled' in de state van App aangemaakt.

Use case 2: Tonen foutstatus

Het leek mij het meest duidelijk om een aparte ErrorButton component aan te maken, zodat ik drie buttons in de interface kon plaatsen voor het triggeren van de errors.
De onClick prop van de ErrorButton component verwijst naar de triggerError prop in App, die op zijn beurt weer verwijst naar de functies triggerError1, triggerError2 of triggerError3 in App. Deze veroorzaken het volgende gedrag:

- De status message in de interface verandert in "ERROR"
- De state van App wordt aangepast: de interface is disabled en de boolean die regelt dat de error overlay wordt getoond wordt op true geset.
- O.b.v. de error die wordt getriggered, wordt aan de error overlay de specifieke bijbehorende tekst meegegeven.
- De errorState van de mock wordt veranderd in het specifieke cijfer-id van de error
- De bijbehorende triggerError functie van de mock wordt gecalled, wat in dit geval betekent dat er een time-out wordt getriggered. Hierna wordt de callback functie setToReady van App gecalled die de interface weer reset, waardoor de error overlay weer verdwijnt en de interface weer enabled is.
  Met een forceUpdate forceer bij deze reset het re-renderen van de App, zodat ik zeker weet dat App er weet van heeft dat de error state weer op nul is gezet.

Testing

Ik gebruik Enzyme i.c.m. Jest, omdat deze combinatie meer testmogelijkheden biedt dan alleen Jest.
Zo simplificeert de shallow rendering optie van Enzyme het testproces.