Verantwoording

Web browser: De web applicatie is in google Chrome getest op een 1280 x 800 formaat gemaakt wat een 10,1”scherm nabootst.

Voorbeeld Video React Opdracht :
https://youtu.be/lku8AJWg1qg

Use case 1: Selecteren van drank
Componenten: CoffeeButtons.js, MachineStatusText.js
Main: App.js

Doel van de use case 1
Door het kiezen van een drank wordt het mogelijk voor de gebruiker om te kiezen tussen de verschillende soorten koffie, thee en chocolademelk die de machine kan maken.

Hoofdscenario

Component: MachineStatusText.js
Main: App.js

1. De klant drukt op een van de knoppen die een drankje maken.

Zodra de gebruiker op de knop van de koffiemachine drukt wordt in de code van App.js op de lijnen 77 – 103 één van de koffies gekozen. De renderswitch methode zorgt ervoor dat de gekozen namen uit Coffeebutton.js op lijn 20 door middel van de onclick functie worden verkregen. Voor een Switch statement is gekozen omdat het overzichtelijk is en er makkelijk een actie aan kan worden verbonden. Zodra de gebruiker op de button klikt wordt er een check uitgevoerd of als de desbetreffende koffie beschikbaar is door middel van TRUE terug te krijgen van uit de SweetCoffeeMock API. Na de check wordt code this.MachineMakesCoffee() gestart en begint het proces van het maken van de gewenste koffie.

2. Alle user interface elementen worden gedisabled (onmogelijk gemaakt om te gebruiken).

Zodra this.MachineMakesCoffee() is getriggerd wordt de code van lijn 106 – 112 in App.js gestart. Alle Elementen worden door middel van de code op lijn 107 in App.js gedisabled.

3. Statusveld wordt “Machine maakt <gekozen drank>”, gekozen drank wordt vervangen met de naam van de gekozen drank.

In MachineStatusText.js van lijn 3 tot 9 wordt de machinestatusText gegenereerd. Het component ontvangt de properties van de updateMachinetekst en coffeename.

In MachineStatusText.js code wordt er eerst gekeken of als er een waarde in updateMachinetekst zit. Mocht dat zo zijn dan wordt die waarde getoond anders wordt er gekeken of als de coffeename is meegegeven en wordt die waarde samen met Machine maakt

getoond. Mocht er helemaal geen waarde in updateMachinetekst en coffeeName zitten dan wordt “klaar maak keuze” meegegeven.

In het scenario dat de gebruiker bijvoorbeeld de “Americano” button aantikt wordt “Machine Maakt Americano” weergegeven.

4. De machine maakt de gekozen drank.

De setTimeout functie in App.js op lijn 108 doet het maken van de geselecteerde coffee nabootsen.

5. Status geeft weer “Klaar voor keuze”.

In App.js op lijn 156 wordt de machine tekst van het component machinestatusText door middel van setState geupdate. Waardoor de gebruiker op het scherm “Klaar voor keuze ” zal zien aangezien updateMachinetekst nu een waarde heeft.

6. Alle user interface elementen worden enabled (weer ‘aangezet’).

Na 3 seconden van de setTimeout in App.js op lijn 108 wordt de methode machineReadyEnableAllButtons geactiveerd die op de lijnen 134 tot 168 in App.js terug te vinden zijn. Dit geeft als resultaat dat alle knoppen weer beschikbaar zijn.

Alternatief scenario 1

Component: CoffeeSlider.js
Main: App.js

1. Gebruiker stelt niveau van suiker of melk in.

Het CoffeeSlider.js component zorgt voor de view en voor de controle van de sliders.

In CoffeeSlider.js vanaf lijn 50 tot 58 gebeurt het volgende:

- Op lijn 51 en 55 wordt de check uitgevoerd of als sliderValueSugar of sliderValueMilk groter is dan 0.

  Deze keuze heb ik gemaakt om een onderscheidt te maken tussen de Sugar en Milk waarde.
- De lijnen 52 en 56 maken het mogelijk dat de gebruiker de slider kan gebruiken en de waardes op het scherm zichtbaar zijn.
- De lijnen 53 en 57 zorgen elk apart voor een callback naar de lijnen 334 tot 347 in App.js. In de callbacks(handleDataCallBackSugarSlider en handleDataCallBackMilkSlider) wordt elk afzonderlijk de gekozen slidervalue door middel van setState toegewezen. Voor event[0] is gekozen aangezien de benodigde waarde op deze manier kan worden verworven.

2. (Naar hoofdscenario) De klant drukt op een van de knoppen die een drankje maken.
this.MachineMakesCoffee() is getriggerd de code van 106 – 112 gestart. Alle Elementen worden gedisabled in de code van App.js op lijn 107 en de koffie wordt gemaakt.

Alternatief scenario 2

Component: SweetCoffeeMock.js, CoffeeSlider.js
Main: App.js

Preconditie: Melk of suiker is op.
1. Sliders van melk of suiker staan op 0 en zijn disabled en Knop Cappuccino is disabled.
De gebruiker heeft de mogelijkheid om 100 procent van de waarde te gebruiken. Als dat het geval is zullen de slider worden gedisabled aangezien er geen voorraad meer is.

Na het klikken van de gebruiker op 1 van de buttons en slider waarde heeft gekozen en de machine weer alle elementen enabled wordt op lijn 161 in App.js de methode checkSugarAndMilkValueAPI aangeroepen.

Hierin worden de slider values van Milk en Sugar aan de calculateValueSugarAndMilk als parameters doorgegeven.

In SweetCoffeeMock.js op de lijnen 87 tot 101 worden de waardes die vanuit de sliders als parameters worden meegegeven van de voorraad afgehaald. calculateValueSugarAndMilk geeft een promise terug met True als de totale Sugar of Milk value boven 0 is en False als de totale Sugar of Milk value onder 0. Die waarde wordt door middel van setState in de property checkSugarAndMilkValue geplaatst op lijn 177 in App.js. Dit is gedaan aangezien de waarde van checkSugarAndMilkValue later weer gebruikt kan worden.

Mocht de waarde van de methode calculateValueSugarAndMilk uit SweetCoffeeMock.js de promise False geven dan wordt in App.js op lijn 182 de methode DisableCappucinoAndSlider aangeroepen.

De volledige code van de methode DisableCappucinoAndSlider op lijn 237 tot 252 doet het volgende:

- De naam “Cappucino” wordt in de data gezocht en disabled op true gezet.
- De properties van sliderSugarDisabled en sliderMilkDisabled worden op True gezet.

Dit geeft als resultaat dat zodra er geen voorraad meer is de sliders en Cappucino button worden gedisabled.

3. (Naar hoofdscenario) De klant drukt op een van de knoppen die een drankje maken.
Zodra de Sliders en Cappucino button zijn gedisabled heeft de gebruiker nog steeds de mogelijkheid om andere dranken te maken omdat de methode MachineReadyEnableAllButtons de rest van de code afloopt.

Alternatief scenario 3

Component: SweetCoffeeMock.js.
Main: App.js

Preconditie: Chocolade is op.
1. Knoppen van Chocolade en Wiener melange zijn disabled.

De gebruiker heeft in dit scenario de totalchocoladeValue van 10 die in SweetCoffeeMachine.js op lijn 15 is aangegeven op gemaakt. Per druk op de Chocolade button of Wiener Melange wordt de conditional statement op lijn 162 tot 164 in App.js uitgevoerd. Hierin wordt de chocolade Waarde van de Api gecontroleerd op lijn 184 in App.js.

Bij elke druk op de buttons Chocolade of Wiener Melange gaat er 5 af van de totale chocolade waarde. Dit is op lijn 16 in SweetCoffeeMachine.js aangegeven.

Op lijn 103 tot 117 in SweetCoffeeMock.js wordt de methode calculateChocolateValue uitgevoerd. Mocht de totalechocolade waarde lager dan 0 zijn dan wordt geeft promise False terug of mocht totalechocolade boven 0 zijn wordt True terug gegeven.

Mocht promise false terug geven dan wordt in App.js op lijn 187 de methode DisableChocoladeAndWiener uitgevoerd.

De methode disabled de Chocolade en Wiener Melange button.

2. (Naar hoofdscenario) De klant drukt op een van de knoppen die een drankje maken.
Zodra de Chocolade en Wiener Melange buttons zijn gedisabled heeft de gebruiker nog steeds de mogelijkheid om andere dranken te maken omdat de methode MachineReadyEnableAllButtons de rest van de code afloopt.

Scenario 2 en 3 gelijktijdig.

De methode MachineReadyEnableAllButtons voert op lijn 141 de conditional statement uit om te checken of als de checkChocolateValue property op true of false staat. Mocht de waarde van checkChocolateValue op false staan dan worden de buttons Chocolade en Wiener Melange gedisabled.

De waarde van de property checkChocolateValue wordt in de methode checkChocoValueAPI bepaalt te vinden op lijn 187 in App.js . Elke keer wanneer de methode MachineReadyEnableAllButtons wordt gestart worden de bovenstaande stappen uitgevoerd en gecheckt.

Het zelfde scenario is ook op lijn 146 in App.js de conditional statement doet een check of als de checkSugarAndMilkValue property op true of false staat. Mocht de waarde van checkSugarAndMilkValueop false staan dan wordt de button Cappucino gedisabled.

Elke keer wanneer de methode MachineReadyEnableAllButtons wordt gestart worden de bovenstaande stappen uitgevoerd.

Doordat de buttons in het begin van de code in App.js op lijn 141 tot 150 de check uitvoert kunnen beide scenarios tegerlijk plaats vinden. Hiervoor is gekozen omdat op lijn 155 de property coffeeItems door middel van setstate wordt geupdate. Op deze manier worden alle buttons in 1 keer geupdate en mocht 1 button bijvoorbeeld Cappucino tijdens het maken op disabled staan wordt er standaard een check uitgevoerd of als de button nog op disabled staat en niet aanspringt tijdens het maken van de andere koffie.

De Sliders worden in App.js op lijn 161 gecontroleerd en gedisabled. Door middel van deze implementatie zijn beide scenario's mogelijk.

Use case 2: Tonen foutstatus

Doel van de use case 2
Het doel van dit scenario is om als er een onoverkomelijke fout ontstaat in de besturingscomputer van de koffiemachine, het kiezen van dranken (use case 1) niet meer mogelijk wordt gemaakt.

Hoofdscenario

1. De besturingscomputer verstuurd het event “error”.

Vanuit SweetCoffeeMock.js op lijn 119 tot 133 wordt de water druk gecontroleerd.

Als de water druk onder 0 is geeft de promise een reject terug met de tekst error. Mocht de waterdruk boven 0 zijn dan geeft de promise een empty resolve terug aangezien ik alleen geïnteresseerd ben in de error message.

2. Alle user interface elementen worden disabled - Een foutmelding overlay wordt getoont - De storingscode wordt vervangen door een gebruiksmelding.

Als de gebruiker één van de koffies kiest en de machine is klaar met het maken van de koffie wordt in de methode van MachineReadyEnableAllButtons in App.js op de lijnen 165 en 166 de waterdruk en temperatuur waarde van de Api opgevraagd. Mocht de waarde false zijn dan wordt de methode op lijn 208 in App.js geactiveerd. Deze methode zorgt voor het volgende:

- MachineInError methode aanroepen om de knoppen te disabelen en de machine tekst op “error” te plaatsen;
- Gebruikers melding genereren met het juiste meldingsnummer;
- Interne melding generen;
- Door de machineInError propertie wordt een pop up voor de gebruiker gegeneerd met de desbetreffende gebruikersmelding;
- SliderMilk en SliderSugar disabelen.

De bovenstaande acties gelden ook voor wanneer de temperatuur te laag is.

Alternatief scenario 1

Preconditie: Na een storing wordt de storingscode 0 (geen storing) opgestuurd.

1. Foutmelding overlay wordt verwijderd - Alle user interface elementen worden enabled.

Na drie seconden wordt de errorResolved methode vanuit SweetCoffeeMock.js aangeroepen. Als voorbeeld op lijn 198 in App.js wordt de tekst geen water als property meegegeven. De errorResolved methode op lijn 152 tot 172 in SweetCoffeeMock.js zal aan de hand van de aangegeven property(“geen water” of “Temperatuur te laag”) de voorraad van alle waardes weer naar normaal brengen. In dit geval heb ik alleen voor een resolve gekozen aangezien er

verder niks anders moet worden gecheckt. Zodra de promise resolved terug geeft wordt op lijn 198 door middel van then de errorResolved methode in App.js aangeroepen. De errorResolved Methode doet het volgende:

- MachineErrorResolved aanroepen om de knoppen weer te enabelen en de machine tekst op “Klaar maak keuze” te plaatsen;
- De interne melding “Geen storing” wordt gegenereerd;
- SliderMilk, Sugar, Chocolade waarde en Sugar and milk value worden allemaal op true gezet door zijn weer enabled;
- MachineError wordt op false gezet wat er voor zorgt dat de pop up voor de gebruiker verdwijnt.

3. Use case 1: Selecteren van drank wordt gestart.

De gebruiker heeft na alle boven genoemde acties weer de mogelijkheid om de drank te selecteren.