Weer Kompas

Verantwoordingsdocument

Door: Timo Noordzee
Studentnummer: 800010184
Onderwijsinstelling: Novi Hogeschool
Opleiding: HBO ICT Software Development
Leerlijn: Frontend
Inleverdatum: 07-06-2022

Inhoudsopgave

1. Clean code

Het eerste wat ik gedaan heb na het aanmaken van het project is het installeren en configureren van ESLint. ESLint help bij het schijven van clean code door een vaste code style te hanteren in alle javascript bestanden en een aantal vaste best practices verplicht te stellen. Een voorbeeld hiervan is bijvoorbeeld dat er geen unused variables of imports mogen zijn. Vooral tijdens het refactoren komt het snel voor dat oude imports blijven staan. ESLint helpt dit detecteren en oplossen.

Naast het installeren van ESLint heb ik in jsconfig.json de src map als baseUrl gezet om absolute imports te gebruiken in het project. De import `components/common/Button` is namelijk veel duidelijker dan bijvoorbeeld `../../components/common/Button`.

2. Firebase

Na het configureren van ESLint en absolute imports heb ik Firebase toegevoegd aan het project. In eerste instantie had ik `firebase.js` in de src map geplaats, maar dit zorgde voor een import conflict i.c.m. met de absolute paden.

De code `import {firebaseAuth} from “firebase”` kan namelijk op 2 manieren geïnterpreteerd worden.

1. Importeer firebaseAuth van de firebase dependency
2. Importeer firebaseAuth van het bestand `firebase.js` in de src map (absolute import)

Om dit conflict te voorkomen heb ik besloten om `firebase.js` te verplaatsen naar de services map `src/services/firebase.js`.

3. Viewport height

In de applicatie moet de div met de class `app` altijd dezelfde hoogte hebben als de viewport. Om dit te realiseren heb ik in eerste instantie `height: 100vh` gebruikt, wat goed werkt in Chrome en de “Device Mode” emulator in de Chrome DevTools. Toen ik de applicatie echter op een Android toestel testte kwam ik al snel problemen tegen met de hoogte van de viewport. De onderkant van de pagina viel buiten de viewport, maar door verder omlaag te scrollen kon ik de applicatie fullscreen maken en was ook de onderkant van de pagina zichtbaar.

De oorzaak van dit probleem is dat WebKit anders omgaat met `100vh` dan bijvoorbeeld de Chrome desktop browser. Om dit probleem op te lossen maak ik zowel gebruik van `height: 100vh` als van `height: -webkit-fill-available`.

4. Styling, SCSS en variabelen

Voor het stylen van de applicatie heb ik besloten om scss te gebruiken. In eerste instantie vanwege de uitgebreidere syntax dan reguliere css, maar ook met het oog op schaalbaarheid van het project. Vooral bij projecten waar veel van de styling gedeeld wordt tussen componenten kunnen scss mixins het delen van styles tussen componenten vereenvoudigen. In mijn applicatie heb ik vanwege de geringe hoeveelheid styling geen gebruik gemaakt van mixins.

Aangezien de applicatie zowel een donker als een licht thema kent is het niet mogelijk om kleuren hardcoded in de styling te zetten. Dit is echter bij projecten met slechts één thema ook geen goed idee mocht er in de toekomst toch een extra thema toegevoegd moeten worden of als het kleurenpallet wijzigt. Om de 2 verschillende thema's mogelijk te maken zijn er verschillende mogelijkheden die hieronder beschreven staan.

In eerste instantie heb ik overwogen om gebruik te maken van styled components waarbij de styling van een component gezet wordt o.b.v. properties. Door middel van een ThemeProvider (React Context) bovenaan de component tree heeft elke component toegang tot de style hoe diep deze component ook in de tree zit.

Uiteindelijk heb ik besloten om niet voor de styles components oplossing te gaan, maar voor css variabelen. Voor alle kleuren in de applicatie zijn variabelen aangemaakt. Een deel van deze kleuren wordt overschreven in het donkere thema door de `[data-theme="dark"]` selector. Door gebruik kan de maken van css variabelen kunnen de kleuren in elk ander `.scss` bestand gebruikt worden zonder de extra boilerplate code van de styled components oplossing.

5. Navigatie en Authenticatie

Een groot deel van de applicatie is alleen beschikbaar voor ingelogde gebruikers. Alleen de loginpagina is voor elke gebruiker toegankelijk. Voor navigatie maak ik gebruik van de `react-router-dom` v6 dependency simpelweg omdat dit de meest voor de hand liggende optie is.

Voor de pagina's die alleen beschikbaar zijn voor ingelogde gebruikers dient steeds dezelfde check gedaan te worden. Is de gebruiker ingelogd, laat dan de pagina zien. Zo niet, redirect naar de loginpagina. Omdat dit gedeelte logica is heb ik een `AuthenticatedRoute` wrapper component gemaakt die deze logica bevat. Elke pagina kan gewrapt worden in een `AuthenticatedRoute` om deze alleen beschikbaar te maken voor ingelogde gebruikers. Als de gebruiker niet is ingelogd wordt `<Navigate/>` gebruikt met in de state de huidige URL. Na het inloggen wordt de state weer uitgelezen zodat de gebruiker weer teruggestuurd kan worden naar de vorige pagina.

6. AuthContext

Alle logica wat betreft inloggen, aanmelden en uitloggen zit in de `AuthContext`. De `AuthContext` maakt 3 async functies beschikbaar voor andere components, namelijk `signIn`, `signUp` en `SignOut`. Naast deze 3 functies bevat de `AuthContext` een eigen error classes. Zo is voor een e-mailadres dat al in gebruik is de `EmailAlreadyInUseError` gemaakt.

Doordat alle authenticatie logica in de `AuthContext` zit en voor fouten een eigen error class bestaat heeft de `LoginPage` component geen dependency op Firebase Authentication. Stel dat in plaats van Firebase Authentication gekozen wordt voor een andere authenticatie provider hoeft alleen de `AuthContext` aangepast te worden en niet de `LoginPage`. Dit komt de onderhoudbaarheid van de code ten goede doordat components meer loosely-coupled zijn.

7. OpenWeather API token

API tokens moeten over het algemeen gezien worden als gevoelige informatie en verborgen worden voor de gebruiker. Stel dat een kwaadwillende beschikt over de OpenWeather API token kunnen requests gemaakt worden buiten de weer applicatie om. In het geval van de OpenWeather API token is dit niet per se een heel groot probleem aangezien het om een gratis token gaat die eenvoudig in te trekken is. Toch heb ik besloten de token volledig buiten de applicatie te houden.

Om de OpenWeather API token te verbergen heb ik zelf een wrapper API gemaakt in de vorm van een Firebase Cloud Function. Deze API gebruikt de JWT van de ingelogde gebruiker om te authentiseren. Uiteraard kan een kwaadwillende nog steeds de OpenWeather API aanroepen buiten de applicatie om, maar er is een verschil. De JWT is slechts één uur geldig in tegenstelling tot de OpenWeather API token die geldig is tot deze ingetrokken wordt. Daarnaast is het eenvoudiger om slechts één kwaadwillende gebruiker te blokkeren i.p.v. het token te vervangen.

8. Weer iconen

Voor iconen in de app maakt ik gebruik van de `react-icons` package. Deze package maakt gebruikt van een HTML svg element voor het renderen van icons. In principe werkt dit altijd goed, maar voor één specifieke feature is dit een probleem. Bij het bekijken van het weer op een specifieke locatie moet de favicon worden aangepast naar het bijpassende icoon voor het weer.

Om dit te realiseren heb ik alle weer iconen als svg bestanden in de `public/assets/icons` map geplaatst. Middels een effect in de `LocationDetailsPage` zet ik de `href` waarde van de favicon link naar het pad voor het gewenste weer icoon. De `WeatherIcon` component maakt ook gebruik van de svg icons in `public/assets/icons`.

9. useInterval hook

Een feature die vaker voorkomt in de applicatie is het periodiek updaten van de state van een component. Om dit te realiseren heb ik een `useInterval` hook gemaakt. Deze hook gebruikt de `useRef` hook om de callback functie op te slaan als referentie zodat in `setInterval` altijd de meest recente callback gebruikt kan worden.

10. DaytimeIndicator

De applicatie laat de resterende tijd tot zonsondergang zowel als tekst zien, als in een soort progress bar. In tegenstelling tot een standaard progress bar is dit geen rechte lijn, maar een halve circel waarop zich een zon verplaatst. In eerste instantie heb ik gekeken of dit te realiseren was met puur css en dit was het geval. Het lukt om een halve cirkel te maken en over die halve cirkel te verplaatsen. Wat niet eenvoudig lukte was het aanpassen van de kleur links en rechts van de zon.

Als alternatieve oplossing heb ik gekeken naar het genereren van een svg element. Hiervoor heb ik gebruik gemaakt van de `d3-shape` en `d3-scale` dependencies. Deze dependencies maken het eenvoudig om svg elementen te genereren. De svg bestaat uit 3 elementen:

1. Een path voor de volledige halve cirkel
2. Een path voor het gedeelte links van de zon
3. Een svg element voor de zon zelf

Middels de `useInterval` hook update ik de state van de `DaytimeIndicator` elke seconde zodat de countdown timer ook elke seconde de nieuwe waarde toont.

11. Card component

De applicatie kent components die allemaal een soortgelijke opbouw hebben, namelijk de:

1. CurrentWeatherCard
2. HourlyForecastCard
3. DailyForecastCard

Al deze components zitten in een div met dezelfde padding en border-radius. Daarnaast beginnen deze components eerst met een titel en volgt vervolgens de content. Omdat het hier om gedeelte logica gaat heb ik een `Card` component gemaakt waar een titel en children in de props meegegeven kunnen worden. Op deze manier zit de gedeelte logica in één component. Stel dat de layout van een kaart in de app aangepast moet worden hoeft alleen in de `Card` component een aanpassing gemaakt te worden. De aanpassing heeft dan meteen effect op alle components die de `Card` component gebruiken.

12. Accordion

De `Accordion` component is een speciaal soort component. Deze component lijkt op de `Card` component, want ook deze bestaat uit een titel en daaronder de content. Het verschil is echter dat de content in de `Accordion` component in- en uitklapbaar is.

Ik had voor deze component gebruik kunnen maken van het slots design pattern waarbij de titel als html element of component in de properties geplaatst kan worden. Dit verschilt met de `Card` component waar de titel altijd tekst is. Deze aanpak werkt wel, maar komt de leesbaarheid van de code niet ten goede.

Een component library zoals Material UI kent ook een `Accordion` component. Binnen Material UI bestaat een `Accordion` uit 3 delen. De `Accordion` zelf met als children een `AccordionSummary` en `AccordionDetails`. Dit komt te leesbaarheid van de code ten goede aangezien de titel (summary) en content (details) niet als properties van de `Accordion` meegegeven worden, maar als children geplaatst worden.

In mijn applicatie heb ik voor dezelfde oplossing als de Material UI Accordion gekozen. Mijn `Accordion` component loopt door de children en past deze, afhankelijk van of het een `AccordionSummary` of `AccordionDetails` component is, aan middels de `cloneElement` functie.

Om het verschil te herkennen tussen een `AccordionSummary` en `AccordionDetails` component heb ik op de `AccordionSummary` een standaard property met de naam `__TYPE` zitten. De naam `__TYPE` is bewust gekozen om aan te geven dat het hier gaat om een property die niet handmatig aangepast dient te worden. De underscores geven aan dat de property als private gezien moet worden en de hoofdletters dat het om een constante waarde gaat. De `__TYPE` property voor de `AccordionSummary` geeft ik vervolgens de standaard waarde “AccordionSummary”.

De `AccordionSummary` heeft ook een callback functie nodig voor wanneer er geklikt wordt om in- of uit te klappen. Voor deze click handler gebruik ik de property `__on_click`, wederom met underscores om aan te geven dat deze property als private gezien moet worden. In tegenstelling tot `__TYPE` gebruik ik geen hoofdletters, omdat de click handler geen constante waarde is.

De `AccordionDetails` moet in hoogte aangepast worden op basis van of de `Accordion` in- of uitgeklapt is. Om dit te realiseren heb ik in de `Accordion` component een referentie naar het content element in `AccordionDetails`. Voor het doorgeven van de referentie vanuit de `Accordion` naar de `AccordionDetails` component maakt ik gebruik van de `forwardRef` functie. Als ik dit niet doe en een ref property toevoeg in de `cloneElement` functie wordt de referentie op het verkeerde element gezet.

De content in `AccordionDetails` moet alleen onderdeel zijn van de DOM als de `Accordion` uitgeklapt is. In eerste instantie keek ik hierbij alleen naar de opened state en toonde ik de content alleen als deze true was. Dit werkt wel, maar als de `Accordion` dan ingeklapt wordt is de content direct geen onderdeel meer van de DOM terwijl er een transition van 300ms zit op het inklappen van de `Accordion`. De animatie ziet er dan niet goed uit. Om dit te voorkomen heeft de `AccordionDetails` compontent een visible state die 300ms nadat opened op false gezet is ook op false gezet wordt. Hierdoor blijft de content bij het inklappen van de `Accordion` nog 300ms onderdeel van de DOM voordat deze verdwijnt.

12. CSS Grid in LocationDetailsPage

Vrijwel in alle styling maak ik gebruik van de flexbox layout, omdat deze uitermate geschikt is voor het maken van een simpele rij of kolom. Voor de `LocationDetailsPage` component was flexbox echter ongeschikt en was css grid een betere oplossing.

De `Sidenav` component moet op kleine schermen over de pagina heen vallen, maar op grote schermen onderdeel zijn van de pagina. Om dit te realiseren is de `LocationDetailsPage` opgedeeld in een simpel grid met 4 cellen.

header	header
sidenav	content

Standaard wordt voor het content element `grid-area: content-start / sidenav-start / content-end / content-end`. De content neemt dus de volledige onderste rij van de grid in beslag (groene cellen). De sidenav neemt alleen de sidenav cell in beslag en er ontstaat dus een overlap in deze cell. Doordat de `Sidenav` component een hogere z-index heeft en `position:relative` valt deze cover de content heen.

header	header
sidenav	content

Als de viewport width groter is dan 1250px wordt de `grid-area: content` gebruikt voor het content element. De content neemt dus niet langer de volldige onderste rij in beslag, maar alleen de content cell (groene cell). De sidenav en content delen nu hun ruimte in de rij en de sidenav valt niet langer over de content heen.

header	header
sidenav	content

13. Limitaties en doorontwikkeling

De huidige versie van de applicatie kent zijn limitaties en er is uiteraard voldoende ruimte voor doorontwikkeling. Een aantal limitaties en mogelijke doorontwikkelingen staan hieronder beschreven. Deze doorontwikkklingen zijn in deze versie niet meegenomen omdat deze buiten de opgestelde lijst van requirements vallen.

13.1 Account beheren

Momenteel is het alleen mogelijk om een account aan te maken en in te loggen. De gebruiker heeft in deze versie niet de mogelijkheid om het wachtwoord te wijzingen of het account te verwijderen. Aangezien Firebase authentication als authenticatie provider gebruikt wordt zijn deze functionaliteiten eenvoudig toe te voegen. Er zou een nieuwe pagina “Mijn account“ moeten komen waarna de gebruiker vanuit de `Sidenav` kan navigeren. Deze pagina bevat dan een formulier om het wachtwoord aan te passen. Middels een simpele API aanroep naar een cloud function kan het wachtwoord gewijzigd worden. Ook het verwijderen van het account kan middels een cloud function API waarbij het bijbehorende user document in Firestore ook automatisch verwijderd wordt.

13.2 Ranschikking in de toekomst bekijken

De huidige rangschikking pagina kijkt naar de weersvoorspelling per uur en pakt de eerste entry. De rangschikking is dus altijd gebaseerd op de weersvoorspelling in dat specifieke uur. Mogelijk wil de gebruiker het weer in verschillende plaatsen niet voor het huidige tijdstip vergelijken, maar het weer over 4 uur. In de `RankingPage` component moet `const weather = location.weather.hourly[0]` dan aangepast worden in `const weather = location.weather.hourly[4]`. De index die gebruikt wordt in `location.weather.hourly` moet dan dus variabel worden. Aan de `RankingPage` component zou een slider toegevoegd kunnen worden waarmee de gebruiker “door de tijd kan schuiven”.

13.3 Meer instellingen voorkeursweer

De gebruiker heeft momenteel wel invloed op hoe het weerscijfer berekend wordt, maar invloed is nog beperkt. De gebruiker kan een voorkeur instellen voor 4 categorieën (temperatuur, zon, wind en regen), maar die wegen momenteel allemaal even zwaar. Het kan zijn dat de hoeveelheid zon helemaal niet relevant is voor de gebruiker, maar de kans op regen en windschaal juist wel heel belangrijk. Om de gebruiker meer controle te geven over de berekening van het weerscijfer zou de gebruiker een weging moeten kunnen geven aan de verschillende categorieën. Daarnaast zouden er nieuwe categorieën zoals hoeveelheid bewolking en zichtafstand toegevoegd kunnen worden.

13.4 Presets voor voorkeursweer

Momenteel kan de gebruiker slechts één soort voorkeursweer instellen. Het is alleen niet logisch om in de zomer en in de winter exact dezelfde voorkeursweer te gebruiken. Daarnaast kan het bijvoorbeeld zo zijn dat de gebruiker in de zomer een andere weersvoorkeur heeft voor een BBQ dan voor tijdens het surfen. Om het voor de gebruiker makkelijker te maken om te wisselen tussen voorkeursweer zou er de mogelijkheid moeten komen om de weersvoorkeur instellingen op te slaan als preset. Op de `RankingPage` zou dan een dropdown moeten komen om te wisselen tussen de verschillende presets.