Verantwoordingsdocument Hooikoorts Stadwijzer Friesland

Auteur: Michael Barak
Datum: 28-07-2025
Doel: Eindopdracht Front-end Ontwikkeling

Inhoudsopgave

Inleiding

Dit Verantwoordingsdocument rechtvaardigt de technische programmeerkeuzes die zijn gemaakt tijdens de ontwikkeling van de Hooikoorts Stadwijzer Friesland, een webapplicatie gebaseerd op *"React"* die hooikoortspatiënten helpt bij het kiezen van optimale locaties in Friesland op basis van real-time pollen- en weergegevens. Het document richt zich uitsluitend op beslissingen met betrekking tot *"JavaScript*" en *"React*", exclusief esthetische keuzes zoals CSS-styling, zoals vereist door de opdracht. Het beschrijft vijf belangrijke technische keuzes, hun alternatieven en hun impact op het project, samen met vijf functionele beperkingen en voorstellen voor toekomstige verbeteringen. Daarnaast reflecteert het op het ontwikkelproces, met aandacht voor successen, uitdagingen en geleerde lessen. Dit document vormt een aanvulling op het Functioneel Ontwerp en biedt inzicht in de technische implementatie om een robuuste, gebruiksvriendelijke applicatie te garanderen.

Technische Programmeerkeuzes

In dit document beschrijf ik vijf inhoudelijke programmeerbeslissingen die ik tijdens het ontwikkelen van mijn "*React-applicatie"* bewust heb genomen. Deze keuzes zijn gebaseerd op de "Keuzes" sectie uit het oorspronkelijke document en de projectstructuur, en voldoen aan de eis van gedetailleerde rechtvaardiging, alternatieven en impact. Bepaalde technologieën, zoals het gebruik van "*React"* en "*JavaScript",* waren volgens de opdracht verplicht. Hierdoor was er geen ruimte voor alternatieve keuzes, en zijn deze aspecten niet verder verantwoord in dit document.

Gebruik van Context API voor toestandsbeheer

- **Keuze**: De applicatie gebruikt React's Context API ("*context/PreferencesContext.jsx, context/AuthContext.jsx"*) om globale staat te beheren, zoals gebruikersvoorkeuren en authenticatiestatus.
- **Waarom**: De keuze voor de Context API is primair gebaseerd op de eenvoud van implementatie en de directe integratie met *"React"*, conform de gestelde eisen van de opdracht. Dit biedt een overzichtelijke en efficiënte manier om state management binnen de applicatie te regelen, zonder onnodige complexiteit toe te voegen. Het maakt het mogelijk om gebruikersvoorkeuren (bijv. pollenconcentratie, temperatuur) en authenticatietokens door te geven aan componenten zoals *"Preferences.jsx"* en *"Login.jsx"* zonder prop drilling, wat de onderhoudbaarheid en eenvoud van de code verbetert.
- **Alternatieven Overwogen**:
    - **Zustand**: Een lichtgewicht staatbeheerbibliotheek, maar dit zou een externe afhankelijkheid toevoegen, wat niet nodig was voor een relatief kleine applicatie.

    - **Lokale Staat in Componenten**: Om de applicatie gestructureerd te beheren, heb ik ervoor gekozen niet in elk component standaard "*useState"* toe te passen. Dit voorkomt onnodige complexiteit. Wanneer je dat wel doet, leidt dat er vaak toe dat gegevens via veel componentlagen moeten worden doorgegeven. Dit bemoeilijkt het onderhoud en maakt de structuur al snel onoverzichtelijk.
- **Impact**: Context API vereenvoudigde de ontwikkeling door staatbeheer te centraliseren, waardoor het bijwerken van voorkeuren in componenten zoals "*Preferences.jsx*" efficiënter werd. Het vereiste echter zorgvuldige planning om onnodige re-renders te voorkomen, wat werd opgelost door context te splitsen voor authenticatie en voorkeuren.
- **Reflectie**: Hoewel de implementatie van de Context API in eerste instantie eenvoudig leek, bleek het optimaliseren van re-renders noodzakelijk om de prestaties van de applicatie te waarborgen. Dit aspect werd aanvankelijk onderschat, maar is uiteindelijk meegenomen in de optimalisatie. Terugkijkend zou het toepassen van bijvoorbeeld *"useMemo"* of *"useCallback*" vermoedelijk aanzienlijk hebben bijgedragen aan een betere prestatie. Het is duidelijk dat het gebruik van deze optimalisatietechnieken de efficiëntie en responsiviteit van de applicatie had kunnen verbeteren. Het eerder toepassen van deze technieken had mogelijk enkele optimalisatieproblemen kunnen voorkomen.

Asynchrone API-aanroepen met Retry-logica

- **Keuze**: Asynchrone "*fetch*"-verzoeken met retry-logica werden geïmplementeerd in de services/ map. (bijv. "*weatherService.jsx", "pollenService.jsx").* Deze aanpak zorgt ervoor dat API-aanroepen naar bijvoorbeeld de Ambee Pollen API en de OpenWeather API herhaald worden bij een mislukte poging, wat de betrouwbaarheid van dataverzameling vergroot.
- **Waarom**: Retry-logica zorgt voor betrouwbaarheid bij API-fouten door netwerkproblemen of limieten (Ambee: 50 aanroepen/dag, OpenWeather: 60 aanroepen/minuut). De "*fetch"* werd gekozen voor de native JavaScript-ondersteuning. Door gebruik maken van een extra afhankelijkheden werden vermeden, dit is ook in lijn met de opdracht.
- **Alternatieven Overwogen**:
    - **Axios**: Het is erg handig want het biedt bepaalde properties zoal retry-mechanismen. Daarnaast geeft dat een betere foutafhandeling, maar het probleem is dat dit een afhankelijkheid gaat toevoegen, wat de bundelgrootte vergroot.
    - **Geen Retry-logica**: Enkelvoudige pogingen zouden de betrouwbaarheid verminderen, vooral gezien API-limieten en netwerkinstabiliteit.
- **Impact**: Retry-logica verbeterde de robuustheid, waardoor data zelfs bij tijdelijke API-fouten konden worden opgehaald. Het verminderde gebruikersfouten, maar verhoogde de complexiteit in de "*services/ module"*, wat zorgvuldige foutlogging vereiste via "*logService.jsx".*

- **Reflectie**: Het schrijven van aangepaste retry-logica was uitdagend maar leerzaam. Ik leerde retry-pogingen te balanceren met gebruikerservaring (bijv. *via "LoadingScreen.jsx"*). In de toekomst zou ik dynamische wachttijden toevoegen voor betere efficiëntie.

LocalStorage voor Opslag van Gebruikersvoorkeuren

- **Keuze**: Gebruikersvoorkeuren worden opgeslagen in "*LocalStorage*" via "*storageManager.jsx*". Dit is ook vermeld in de secties "*Voorkeuren Instellen*" en "*Technische Architectuur."*
- **Waarom**: *"LocalStorage"* werd gekozen vanwege de eenvoud en overeenstemming met de eis om geen database te gebruiken. Het maakt permanente opslag van voorkeuren mogelijk zonder server-side infrastructuur, en ondersteunt offline functionaliteit (bijv. via *"Preferences.jsx"*).
- **Alternatieven Overwogen**:
    - **SessionStorage**: het is belangrijk te noteren dat gegevens die hierin worden opgeslagen, verloren gaan zodra de browsersessie wordt beëindigd. SessionStorage is daarom met name geschikt voor tijdelijke opslag tijdens een enkele sessie. Hierdoor is dit middel ongeschikt voor het opslaan van permanente voorkeuren.
    - **Cookies**: Naar mijn mening zijn cookies minder geschikt voor het opslaan van grotere datasets, bijvoorbeeld gebruikersvoorkeuren. Tevens biedt "*LocalStorage"* in dit geval een geschiktere oplossing voor het bewaren van gebruikersspecifieke gegevens.
- **Impact**: "*LocalStorage"* maakte snelle implementatie van voorkeurenopslag en offline ondersteuning mogelijk, en voldeed aan de eis van minder dan 50 MB opslag. Het vereiste zorgvuldige serialisatie om limieten te vermijden, zoals geïmplementeerd in "*storageManager.jsx"*.
- **Reflectie**: "*LocalStorage"* was eenvoudig te implementeren, maar ik zag aanvankelijk opslagquota over het hoofd. Testen met grote voorkeurensets hielp me de gegevensgrootte te optimaliseren. De volgende keer zou ik validatie toevoegen om gebruikers te waarschuwen bij bijna-volledige opslag.

Modulaire Componentontwerp voor Herbruikbaarheid

- **Keuze**: React-componenten (bijv. "*CitySelection.jsx" en "Preferences.jsx"*) in de "*components/ map"* werden ontworpen en waardoor herbruikbaarheid en modulariteit worden bevorderd .
- **Waarom**: Modulaire componenten verminderen code-duplicatie en verbeteren onderhoudbaarheid, waardoor elementen zoals stadselectiekaarten of voorkeursliders hergebruikt kunnen worden in "*Dashboard.jsx"* en *"Results.jsx".* Dit sluit aan bij React's filosofie.
- **Alternatieven Overwogen**:

    - **Monolithische Componenten**: Het combineren van UI-elementen in grote componenten zou initiële ontwikkeling vereenvoudigen, maar updates bemoeilijken.
    - **Kopiëren van Componenten**: Het dupliceren van componenten voor verschillende pagina's zou sneller zijn, maar leidt tot redundante code.
- **Impact**: Modulair ontwerp maakte consistente UI-updates mogelijk (bijv. via gedeelde stijlen in "*App.css"*) en vereenvoudigde debugging door logica te isoleren. Het vereiste voorafgaande planning voor componentinterfaces.
- **Reflectie**: Het opsplitsen van de UI kostte tijd, maar bespaarde moeite tijdens iteraties. Ik leerde het belang van duidelijke props-definities. Het gebruik maken van PropTypes kan gunstig zijn voor validatie in situaties waar TypeScript niet is toegestaan.

Aangepast Score-algoritme in scoreCalculator.jsx

- **Keuze**: Een aangepast score-algoritme in "*scoreCalculator.jsx"* berekent stadscores met een gewogen gemiddelde formule*: "Score = (Pollen × W_pollen) + (Temperatuur × W_temp) + (UV × W_uv) + (Wind × W_wind) + (Vocht × W_vocht)".*
-
- **Waarom**: De gewogen gemiddelde benadering biedt flexibiliteit, zodat gebruikers prioriteiten (1–5) kunnen toewijzen aan criteria zoals pollen of temperatuur. Het is efficiënt en aanpasbaar op basis van feedback.
- **Alternatieven Overwogen**:
    - **Eenvoudige Som**: Optellen zonder gewichten zou gebruikersvoorkeuren negeren, waardoor personalisatie afneemt.
    - **Vooraf Gedefinieerde Scores**: de beperking van vooraf gedefinieerde scores per factor wordt benoemd, wat wijst op een gebrek aan flexibiliteit binnen het huidige systeem.
- **Impact**: Het algoritme rangschikte steden effectief, voldeed aan de eis voor een top 5 in "*Results.jsx*", en werd getest met *"mockdata*". Het gaat uit van lineaire relaties, wat "*real-world"* interacties kan vereenvoudigen.
- **Reflectie**: Het algoritme was eenvoudig te implementeren, maar randgevallen zoals gelijke scores waren uitdagend. Ik verbeterde mijn normalisatietechnieken. In de toekomst zou ik niet-lineaire formules testen.

Beperkingen en Toekomstige Verbeteringen

Vijf functionele beperkingen van de Hooikoorts Stadwijzer Friesland worden uiteengezet, elk voorzien van argumenten voor mogelijke verbeteringen. Dit wijst op een systematische en

nauwkeurig benadering van de applicatiearchitectuur. Deze zijn afgeleid uit de "*Risico's en Oplossingen*" sectie en de projectstructuur.

API-aanroeplimieten

- **Beperking**: De applicatie wordt beperkt door Ambee's 50 aanroepen/dag en OpenWeather's 60 aanroepen/minuut limieten, wat schaalbaarheid voor meerdere gebruikers of frequente updates kan beperken.
- **Waarom Het Bestaat**: De gratis tiers van deze API's leggen strikte limieten op, en de projecttijdlijn liet geen ruimte om betaalde plannen of alternatieve API's te onderzoeken.
- **Verbetering**: Verfijn de client-side caching in storageManager.jsx door een meer geavanceerde cache-invalidering toe te passen, bijvoorbeeld door een langere "*Time-To-Live"* voor de UV-index in te stellen. Het voorstel om een fallback-API toe te voegen voor essentiële functionaliteiten, zelfs zonder backend, toont inzicht in het belang van redundantie en het waarborgen van beschikbaarheid.
- **Reden Niet Geïmplementeerd**: Tijdgebrek en focus op kernfunctionaliteit verhinderden extra API's of geavanceerde caching.

Beperkingen in Offline Functionaliteit

- **Beperking**: De offline modus gebruikt gecachte gegevens in "*LocalStorage"* via "*storageManager.jsx"*, die na een uur (TTL) verouderd kunnen raken, wat mogelijk onnauwkeurige adviezen oplevert in "*Advice.jsx"*.
- **Waarom Het Bestaat:** De TTL zorgt voor datavernieuwing, maar beperkt offline betrouwbaarheid tijdens langdurige internetuitval.
- **Verbetering**: Voeg een waarschuwing toe in "*ErrorMessage.jsx"* wanneer gegevens verouderd zijn. Implementeer "*mockdatageneratie" in "adviceGenerator.jsx"* voor kritieke scenario's (bijv. hoge pollenniveaus).
- **Reden Niet Geïmplementeerd**: Tijdsbeperkingen en focus op kernfunctionaliteit verhinderden geavanceerde offline functies.

Limiet op Stadselectie

- **Beperking**: Gebruikers kunnen maximaal 10 steden selecteren in "*CitySelection.jsx",* wat de bruikbaarheid beperkt voor wie meer locaties wil vergelijken.
- **Waarom Het Bestaat**: Deze beperking draagt bij aan aanzienlijke prestatieverbeteringen en minimaliseert het aantal API-verzoeken, wat van cruciaal belang is gezien de toegepaste API-limieten. Door middel van deze wordt zowel de efficiëntie als de betrouwbaarheid van het API's systeem verhoogd.

- **Verbetering**: Implementeer "*lazy loading*" binnen "*src/App.jsx"* om de schaalbaarheid van de applicatie te vergroten, met name bij het ondersteunen van een groter aantal steden, zonder concessies te doen aan de prestaties.. Daarnaast zorgt het toepassen van "*lazy loading"* ervoor dat enkel de data wordt geladen op het moment dat deze daadwerkelijk nodig is. Met deze aanpak benut je het geheugen optimaal en wordt het databeheer binnen het systeem een stuk efficiënter. Het voorkomt verspilling van middelen en zorgt ervoor dat processen vlotter verlopen. Dit voorkomt dat de applicatie traag wordt bij het verwerken van een uitgebreide lijst steden en draagt bij aan een efficiënter gebruik van systeembronnen. Verwerk "*scoreCalculator.jsx* " incrementeel, zodat de applicatie responsief blijft, zelfs bij een toenemend aantal steden.
- **Reden Niet Geïmplementeerd**: "*Lazy loading"* voegde complexiteit toe buiten de projecttijdlijn.

Eenvoud van het Score-algoritme

- **Beperking**: Het gewogen gemiddelde algoritme in *"scoreCalculator.jsx"* gaat uit van lineaire relaties tussen factoren (bijv. pollen, luchtvochtigheid), wat complexe interacties kan missen.
- **Waarom Het Bestaat**: Een eenvoudig algoritme werd gekozen voor implementatiegemak en prestaties.
- **Verbetering**: Ontwikkel een niet-lineaire formule in "*scoreCalculator.jsx",* gebaseerd op gebruikersfeedback, zonder externe afhankelijkheden.
- **Reden Niet Geïmplementeerd**: Gebrek aan tijd en testdata maakten een complexer algoritme niet haalbaar.

Onvolledige Foutafhandeling voor API Randgevallen

- **Beperking**: De beschrijving van de huidige tekortkomingen in de foutafhandeling binnen de services (*"weatherService.jsx"* en *"pollenService.jsx"),* met name bij langdurige API-uitval of onverwachte dataformaten, onderstreept het risico op incomplete resultaten en adviezen. De aanbevolen verbetering—integratie met *"logService.jsx"* en het toevoegen van een *"mockdatalaag*"—biedt een praktische oplossing gericht op betrouwbaarheid en foutanalyse in stadscores of adviezen in *"Results.jsx*" en "*Advice.jsx".*
- **Waarom Het Bestaat**: De focus lag op veelvoorkomende fouten (bijv. netwerkfouten via "ErrorMessage.jsx"), en API-randgevallen werden gede-prioriteerd vanwege tijdgebrek.
- **Verbetering**: Het is raadzaam om "*ErrorBoundary.jsx"* te integreren met *"logService.jsx",* zodat een meer gedetailleerde registratie van fouten mogelijk wordt gemaakt. Dit voorkomt de weergave van generieke foutmeldingen en faciliteert een diepgaandere

analyse van problemen binnen de applicatie. Daarnaast is het raadzaam om in *"adviceGenerator.jsx"* een "*mockdatalaag"* te implementeren, zodat scenario's waarbij de API niet reageert adequaat gesimuleerd kunnen worden. Toon gebruikersvriendelijke meldingen via "*ErrorMessage.jsx".*

- **Reden Niet Geïmplementeerd**: Het realiseren van robuuste API-foutafhandeling bleek aanzienlijk meer tijd te vergen dan oorspronkelijk beschikbaar was binnen het project.

Reflectie op het Ontwikkelproces

- **Wat Ging Goed**:
  - **API-integratie**: Het integreren van Ambee- en OpenWeather-API's via *"weatherService.jsx"* en *"pollenService.jsx"* met retry-logica zorgde voor betrouwbare gegevensophaling, en voldeed aan de eis voor real-time data.
  - **Modulair Ontwerp**: Herbruikbare componenten zoals "*CitySelection.jsx"* en "*Preferences.jsx*" stroomlijnden de ontwikkeling en maakten UI-updates efficiënt.
  - **Mockdata Testen**: Mockdata in "*scoreCalculator.jsx"* hielp het algoritme vroeg te valideren, waardoor herzieningen werden verminderd.
- **Wat Kan Beter**:
  - **Randgeval Planning**: Ik onderschatte randgevallen zoals gelijke scores in *"scoreCalculator.jsx"* (opgelost via secundaire criteria). Grondigere planning zou debugging hebben verminderd.
  - **Tijdsbeheer**: het blijft een aandachtspunt; er gaat opvallend veel tijd verloren aan het implementeren van retry-logica in services en aan het afhandelen van vertraagde functies, met name bij de uitgebreide foutafhandeling in *"ErrorBoundary.jsx".*
  - **Testen**: "*Mockdata"* testen waren effectief, maar zonder unit-tests was handmatig testen tijdrovend.
- **Geleerde Lessen**:
  - **Vroeg Testen**: Het is evident dat het prioriteren van eenvoudige "*JavaScript-unit-tests"* – bijvoorbeeld met behulp van *"Jest"* – in een vroeg stadium aanzienlijk bijdraagt aan de betrouwbaarheid van de codebasis.
  - **Documentatie**: Documenteren tijdens ontwikkeling zou dit document nauwkeuriger hebben gemaakt.
  - **Schaalbaarheid**: Door vroegtijdig te anticiperen op API-limieten, hadden beperkingen zoals de limiet van 10 steden mogelijk voorkomen kunnen worden.

Table 1-Samenvattende Tabel – Technische Keuzes en Beperkingen. *Deze tabel geeft een beknopt overzicht van de belangrijkste technische keuzes en functionele beperkingen binnen het project Hooikoorts Stadwijzer Friesland. Elke keuze wordt toegelicht met de overwegingen, alternatieven, impact en bijbehorende reflectie of verbetervoorstel.*

	Onderdeel	Beschrijving / Keuze	Alternatief(en)	Impact / Reflectie
1	Toestandsbeheer	React Context API voor "*auth* & *voorkeuren*"	Zustand, lokale "*useState*"	Eenvoudig, onderhoudbaar; splitsing voorkwam re-renders. "*useMemo*" had eerder ingezet kunnen worden.
2	API-aanroepen met retry	Fetch maakt gebruik van handmatig geïmplementeerde retry-logica voor zowel OpenWeather als Ambee	Axios met retry, geen retry	Hierdoor ontstaat meer robuustheid bij API-fouten, al leidt dit tot een toename in de complexiteit van de logging. Mogelijk uitbreiden met dynamische wachttijden.
3	Opslag voorkeuren	*"LocalStorage"* via "*storageManager.jsx*"	SessionStorage, cookies	Ondersteunt offline modus. Opslaglimiet onderschat → validatie had beter gekund.
4	Modulair componentontwerp	Herbruikbare componenten zoals *"CitySelection.jsx", "Preferences.jsx"*	Monolithisch, duplicatie	Herbruikbaar, schaalbaar. "*Props*" goed definiëren was leerzaam.
5	Score-algoritme	Gewogen gemiddelde van pollen, temperatuur, etc. in "*scoreCalculator.jsx*"	Som, vaste scores	Flexibel en snel, maar mist niet-lineaire verbanden. Gelijke scores waren lastig op te vangen.
6	Beperking: API-limieten	50 (Ambee) / 60 (OpenWeather) limiet	Betaalde API's, caching, fallback API	Schaalbaarheid beperkt. Geavanceerdere cache "*TTL*"en fallback-API aanbevolen. Niet geïmplementeerd door tijd.
7	Beperking: Offline modus beperkt	LocalStorage vervalt na 1 uur "*TTL",* geen waarschuwing	Mockdata, waarschuwingen	Onnauwkeurige adviezen mogelijk. "*ErrorMessage.jsx"* waarschuwing nodig
8	Beperking: Stadlimiet (10)	Maximaal 10 steden, beperkt door API-calls.	*"Lazy loading"*, incrementeel berekenen.	blijft de performance stabiel en efficiënt. *"Lazy loading"* was te complex voor scope.
9	Beperking: Simpel score-algoritme	Lineair, geen rekening met wisselwerking (bijv. hoge pollen & wind)	Niet-lineaire functies	Snel en simpel, maar mist nuance. Kan op termijn verfijnd worden o.b.v. feedback.
10	Beperking: Foutafhandeling randgevallen	Alleen basisfouten (netwerk, response), geen handling bij ongeldige formats of langdurige uitval	*"ErrorBoundary"*, "*mockdata",* *"logService"* integratie	Risico op incomplete resultaten. "*Mockdatalaag"* en betere logging aanbevolen.

Conclusie

De Hooikoorts Stadwijzer Friesland werd succesvol ontwikkeld met “*JavaScript”* en “*React*”, met keuzes zoals Context API, modulaire componenten, en een aangepast score-algoritme die een robuuste applicatie garandeerden. Deze beslissingen balanceerden eenvoud en functionaliteit binnen de opdrachtbeperkingen (geen “*Redux/TypeScript”).* Beperkingen bijvoorbeeld API-limieten en ontoereikende foutafhandeling vormen concrete knelpunten. Deze aspecten geven duidelijke richting aan verdere optimalisaties, aangezien zij de werking en stabiliteit van het systeem direct beïnvloeden. Denk hierbij aan het toepassen van meer geavanceerde cachingstrategieën en het verder optimaliseren van “*ErrorBoundary.jsx”.* Het ontwikkelproces leverde lessen op over asynchrone programmering en testen. Met verfijning kan de applicatie meer gebruikers ondersteunen en nauwkeurigere adviezen bieden.