Verantwoordingsdocument – SportData App

Inleiding

In dit document verantwoord ik mijn technische keuzes in de SportData-app. Per keuze beschrijf ik het doel, de overwogen alternatieven, wat er eerst misging en hoe ik dat heb opgelost. Daarna volgen beperkingen met realistische vervolgstappen en een korte persoonlijke noot (tijd/kennis/focus). De opbouw sluit aan op het beoordelingsformulier: minstens vijf keuzes met diepgang en reflectie, en minstens vijf beperkingen met argumentatie.

Keuzes (met alternatieven en reflectie)

2) SeasonSelector als los component

Waarom
Seizoenskeuze zou je in de toekomst op meerdere plekken kunnen gebruiken. Ik heb een herbruikbare component gemaakt met props (value, onChange, yearsBack, en keuze om bij huidig of vorig jaar te starten).

Alternatieven

- Inline <select> per pagina (snel maar duplicatie).
- Generieke Select + dunne SeasonSelector (nog herbruikbaarder, iets meer werk).

Eerst fout gegaan
Een hulpfunctie (clsx) aangeroepen zonder import, wat een lint-fout gaf.

Oplossing/reflectie
Vervangen door eenvoudige class.

3)Veilige render na data-fetch (guards i.p.v. “blind” .map())

Waarom
API-antwoorden kunnen falen of leeg zijn. Zonder checks crasht de UI snel (bijv. “Failed to fetch” gevolgd door een .map op undefined). Met render-guards blijft de app stabiel en voorspelbaar.

Alternatieven

- Zonder guards direct .map aanroepen: minder code, maar leidt tot runtime-errors.
- Alleen try/catch in useEffect: vangt het ophalen af, maar niet de **render**.

Wat ging eerst mis
Bij netwerkfouten of onvolledige seizoensdata probeerde de UI tóch te mappen → console-errors en een kapot scherm.

Oplossing & wat ik leerde
Ik werk overal met een vaste state-vorm ({ loading, error, data }) en render-regels:

- Eerst loading tonen,
- dan bij een fout een duidelijke error-melding met “opnieuw proberen”,
- pas mappen als Array.isArray(…) waar is; anders een nette Empty state.
  Dit voorkomt crashes en geeft de gebruiker altijd feedback.

  Volgende stap: een kleine, herbruikbare fetch-helper (of later React Query) zodat dit patroon overal

4) Semantische lijsten

Waarom
Lijsten zoals de competitielijst en favorietenteams anders ingericht. Ik richt ze in als ul/li met items als article.

Alternatieven

- Alles met : werkt, maar minder semantiek en slechtere screenreader-ervaring.

Eerst fout gegaan
Begonnen met div-lijsten, wat minder duidelijke code geeft.

Oplossing/reflectie
Omgezet naar lijsten. Dit sluit aan op de feedback om weg te blijven van “alleen divs”.

5) Consistent thema met CSS-variabelen en zichtbare contrast-fix

Waarom
Kleuren en theming beheer ik met CSS-variabelen, zodat het uiterlijk consistent is en makkelijk aanpasbaar.

Eerst fout gegaan
In de basis-container stond dezelfde kleur voor achtergrond én tekst, waardoor tekst soms verdween.

Oplossing/reflectie
Tekstkleur aangepast naar een lichte tint en een paar “text-on-bg” keuzes vastgezet. Dit soort fouten vang ik voortaan eerder af.

6) Favorieten-UI met compacte vormindicatoren

Waarom
Bij favorieten toon ik recente resultaten als kleine W/G/V/N-blokjes; snel scanbaar zonder veel tekst.

Alternatieven

- Lange tekstregels nemen onnodig ruimte in.

Eerst fout gegaan
Op smalle schermen werd de grid te krap door kolomstroom.

Oplossing/reflectie
Mobiel schakelt de layout naar rijen; in een volgende iteratie maak ik dit nog vloeiender met een auto-fit grid.

Beperkingen (met persoonlijke noot en vervolgstap)

1. *Onvolledige stand per seizoen*
   Beperking: Competities leveren maar 5 teams terug, en sommige geen teams.
   Waarom nu niet opgelost: afhankelijk van externe API; focus lag op kernflow.
   Volgende stap: Betaalde versie nemen waardoor je deze data wel krijgt.

2. *Globale body-stijl verspreid over component-CSS*
   Beperking: body-achtergrond staat in meerdere componentbestanden.
   Waarom nu niet: tijd; het werkte visueel.
   Volgende stap: globale stijlen centraliseren (één plek), component-CSS lokaal houden.

3. *CORS-oplossing alleen in development*
   Beperking: productie heeft nog geen eigen proxy/caching.
   Waarom nu niet: scope en tijd.
   Volgende stap: kleine Node/serverless proxy met CORS-headers en response-cache.

4. *Rate limiting bij veel requests*
   Beperking: free tier wordt snel geraakt bij parallelle calls.
   Waarom nu niet: beperkte tijd/kennis voor een centrale queue.
   Volgende stap: centrale fetch-laag met queue, retry/backoff of overstappen op een data-bibliotheek.

5. *Zoeken/filteren beperkt*
   Beperking: basiszoeken, geen filterchips of sorteerbare kolommen in de stand.
   Waarom nu niet: focus op league→teams→detail.
   Volgende stap: debounced zoeken, filterchips en client-side sortering (of server-side als de API dat ondersteunt).

6. *Formvalidatie basic*
   Beperking: geen schema-validatie (wachtwoordsterkte/regex).
   Waarom nu niet: tijd/kennis.
   Volgende stap: validatieschema's inzetten en veldfouten consistent tonen.

7. *Favorieten-grid kan eenvoudiger*
   Beperking: huidige grid-opzet is wat complex.
   Waarom nu niet: het werkt functioneel; tijd was beperkt.
   Volgende stap: auto-fit grid met minimale kaartbreedte.

8. Thema-contrast moest achteraf gecorrigeerd
   Beperking: basiscontrast was even fout.
   Waarom toen zo: slordigheid/tijdsdruk.
   Volgende stap: vaste tokens voor tekst-op-achtergrond en een snelle visuele contrastcheck.

Reflectie

De grootste uitdagingen kwamen door externe factoren (CORS, rate-limits, onvolledige seizoensdata). Dat dwong me om robuuster te denken over states, guards en fallbacks. Ik heb geleerd om herbruikbare componenten te maken waar het loont (SeasonSelector), UI-states te standaardiseren en API-realiteit zichtbaar te maken richting de gebruiker. Volgende iteratie richt ik me op een eigen proxy/caching-laag, schema-validatie en een centrale fetch-strategie (queue/retry) om de app minder kwetsbaar te maken voor externe issues.