TITEL VAN HET PROJECT: MealMate

Student: Lily Nordland
Opleiding: Frontend eindopdracht (Full-stack boot camp)
Docent / Begeleider: Nova Eeken
Datum van inlevering:

INHOUDSOPGAVE:

1. Inleiding: 2
2. Technische programmeerkeuzes: 2
2.1 Componentstructuur 2
2.2 Routing met BrowserRouter 2
2.3 API-calls rechtstreeks in useEffect 3
2.4 Gebruik van localStorage in plaats van Context 3
2.5 Geen helperfuncties voor error/loading handling 3
3. Limitaties en mogelijke doorontwikkelingen: 4
3.1 Authenticatie en state management 4
3.2 Loading states 4
3.3 Error fallback UI 4
3.4 Formulier validatie 4
3.5 Favorieten functionalities 5
4. Reflectie op leerproces: 5

1. Inleiding:

Dit verantwoordingsdocument beschrijft de technische keuzes die ik heb gemaakt tijdens de ontwikkeling van mijn frontend-applicatie “MealMate / Simple Recipe Maker”. Het doel van dit document is om inzicht te geven in waarom bepaalde oplossingen zijn gekozen binnen JavaScript en React, welke alternatieven overwogen zijn en wat ik ervan geleerd heb. Daarnaast reflecteer ik op de beperkingen van de huidige implementatie en geef ik aan welke doorontwikkelingen mogelijk zijn in een toekomstige versie.

2. Technische programmeerkeuzes:

2.1 Componentstructuur

Ik heb ervoor gekozen om mijn applicatie op te bouwen met een duidelijke scheiding tussen pagina-componenten en herbruikbare UI-componenten. De pagina’s bestaan onder andere uit **LoginPage, SearchPage, ResultsPage, DetailPage, ContactPage en ResetPasswordPage**. Daarnaast heb ik kleinere herbruikbare componenten gemaakt zoals **AppButton, BackButton, CircleImage, RecipeBox, RecipeDetail en SideImages**.

De reden hiervoor is dat deze aanpak zorgt voor overzicht, modulariteit en hergebruik. In plaats van dezelfde knoppen of recepttegels steeds opnieuw te schrijven, kon ik ze één keer definiëren en hergebruiken op meerdere plekken. Een alternatief was om alle logica direct in de pagina's te plaatsen, maar dit zou op de lange termijn leiden tot dubbele code en minder flexibiliteit.

Reflectie: In een toekomstige versie zou ik sommige componenten nog generieker kunnen maken door props uitbreidbaarder te maken, zodat ze in nog meer contexten gebruikt kunnen worden.

2.2 Routing met BrowserRouter

Mijn navigatiestructuur is opgezet met **BrowserRouter, Routes en Route in App.jsx**. Hiermee kan een gebruiker via URL-navigatie naar verschillende delen van de applicatie zoals */search*, */results* en */detail*.

Ik heb bewust gekozen voor **react-router-dom** omdat dit de standaardoplossing is voor Single Page Applications in React. Een alternatief was om conditie-gebaseerde rendering te gebruiken zonder echte URL-routes, maar dit zou de gebruikservaring minder intuïtief maken.

Reflectie: In een latere versie zou ik **Private Routes kunnen toevoegen** zodat bepaalde pagina's alleen toegankelijk zijn voor ingelogde gebruikers.

2.3 API-calls rechtstreeks in useEffect

De data van Spoonacular wordt opgehaald met **fetch rechtstreeks in de useEffect** van *ResultsPage* en *DetailPage*. Dit was de snelste manier om functionele data-integratie te realiseren.

Een alternatief was om de API-logica te verplaatsen naar **losse helperfuncties of een aparte service file**. Dit zou de code loskoppelen van de UI en herbruikbaarder maken.

Reflectie: Ik heb geleerd dat **centrale API-logica de leesbaarheid verbetert**, en dit zou ik in een toekomstige versie willen doorvoeren.

2.4 Gebruik van localStorage in plaats van Context

Omdat authenticatie nog in een basisfase zit, sla ik de **loginstatus op in localStorage** en lees ik deze uit in SearchPage, ResultsPage en DetailPage.

Dit was een **snelle en eenvoudige oplossing** zonder extra complexiteit. Een alternatief was om vanaf het begin **React Context** te implementeren voor globale state.

Reflectie: Voor een kleine applicatie werkte localStorage prima, maar voor schaalbaarheid en beveiliging is **Context of een Token Refresh Flow** noodzakelijk.

2.5 Geen helperfuncties voor error/loading handling

Error- en loading states worden momenteel **inline afgehandeld in dezelfde functies** waar ook de API-call plaatsvindt.

Een professionelere aanpak zou zijn om **een aparte LoadingSpinner-component en ErrorMessage-component** te schrijven.

Reflectie: Dit was een bewuste keuze om eerst werkende functionaliteit te realiseren. Bij de volgende versie wil ik **consistentere UX toevoegen via herbruikbare feedbackcomponenten**.

3. Limitaties en mogelijke doorontwikkelingen:

3.1 Authenticatie en state management

Limitatie: Loginstatus is afhankelijk van localStorage en wordt niet gevalideerd.
Doorontwikkeling: Implementatie van een centrale **AuthContext met Private Routes**.

3.2 Loading states

Limitatie: Tijdens API-calls is er **geen visuele feedback**.
Doorontwikkeling: Een **LoadingSpinner** of skeleton UI toevoegen.

3.3 Error fallback UI

Limitatie: Bij API-fouten wordt geen melding aan de gebruiker getoond.
Doorontwikkeling: Een **ErrorMessage-component** ontwikkelen met retry-knop.

3.4 Formulier validatie

Limitatie: Ingredientvelden en contactformulieren accepteren **lege input**.
Doorontwikkeling: Validatie toevoegen met **handmatige checks of een library zoals Yup**.

3.5 Favorieten functionalities

Limitatie: Gebruikers kunnen **geen recepten opslaan**.
Doorontwikkeling: Een **“Save Recipe” functie** toevoegen met opslag in localStorage of backend.

4. Reflectie op leerproces:

Tijdens dit project heb ik veel geleerd over het opzetten van een React-applicatie met routing en API-integratie. Ik merkte dat ik geneigd was om alles direct in één bestand te schrijven, maar door componenten los te trekken werd mijn code overzichtelijker en herbruikbaarder. Ook realiseerde ik me dat een eerste werkende versie belangrijker is dan alles direct perfect structureren. In de toekomst wil ik **eerder nadenken over centrale state management en betere foutafhandeling**.